

# Pascal NEWSLETTER

NUMBER 3

OREGON SOFTWARE

NOVEMBER 1981

## Memory allocation for Pascal programs

Some Pascal users report that (apparently) small programs will run out of memory on RSX. Often sufficient memory exists, but for various reasons it is not available. To make better use of memory, users should understand how it is allocated and used by Pascal programs.

### Background on RSX

**Task size limits:** Each separate task can address and use at most 32K words (or 64K bytes) of memory, even though the system may have much more main memory. This limit is caused by the PDP-11 16-bit address range. Overlays will not increase this address space, but may allow reuse of some of the space for different purposes.

**Program organization:** The parts of a Pascal program are described under "Run-Time Organization" in the Programmer's Guide of the Pascal-1 or Pascal-2 User Manuals: task header, program code, constants, global variables, tables, stack, and heap. Refer to the run-time section as needed while reading this article.

**Static and dynamic allocation:** Much of a task's memory allocation is static (it does not change over time), set by the Task Builder at task-build time; some memory is dynamic, allocated by the Pascal program at run-time. There are two kinds of dynamic memory allocation: the "stack" and the "heap".

**Dynamic uses of memory — the stack and the heap:** Each time a procedure is called, stack space is reserved for the variables declared in that procedure; when the procedure terminates, or is exited by a **goto**, that stack space is released. When a file is opened, heap space is reserved for the file buffer and file descriptor; each use of the **new** procedure also reserves space on the heap. **Close** and **dispose** release heap space used for files and for **new**, respectively.

### Basic Task Builder operations

Compilation of a program produces a relocatable object module (or modules, if there are external routines). The Task Builder combines the program module(s) with object modules extracted from the Pascal support library (PASLIB) and the system library (SYSLIB). All the ob-

ject modules contain one or more program sections called PSECTs. The Task Builder sorts the PSECTs into alphabetical order, allocating space for the program header, code, constants, and tables of the Pascal program. Pascal-2 tasks contain a PSECT for the global-level variables associated with the program. Pascal-1 tasks do not; instead, global-level variables are allocated when the task is initialized.

The Task Builder also allocates a small stack space in the task image. Pascal programs, however, require a large stack space to hold the local variables associated with each procedure, of which there can be many. The small stack space set up by the Task Builder is used by Pascal tasks only for initialization of the program at run-time.

All Pascal tasks contain a PSECT called **\$\$HEAP**. Normally, the Task Builder does not allocate space to the **\$\$HEAP** PSECT. In this case, the Pascal initialization routine uses the **EXTK\$** (extend task) system directive to expand the task by 2K words; this space is used for the heap and the stack. If the **\$\$HEAP** PSECT has been extended via the **EXTSCT** Task Builder directive, described below, **\$\$HEAP** is used instead for the stack and the heap. In Pascal-1 programs, the global-level variables are then allocated (also by task extension or from **\$\$HEAP**).

Continued on Page 2

## In this issue...

|                                             |         |
|---------------------------------------------|---------|
| RSX memory allocation . . . . .             | Page 1  |
| Reviews . . . . .                           | Page 3  |
| Pascal-2 released for RSTS, RT-11 . . . . . | Page 4  |
| Letters . . . . .                           | Page 5  |
| Information exchange . . . . .              | Page 7  |
| 'For' loop restriction . . . . .            | Page 7  |
| Trouble with trouble reports . . . . .      | Page 8  |
| Pascal for text formatting . . . . .        | Page 9  |
| The Log: Pascal-1 and Pascal-2 . . . . .    | Page 13 |
| RT-11 disk space problems . . . . .         | Page 16 |
| RT-11 Linker problem (again) . . . . .      | Page 16 |
| TSX and TSX-Plus . . . . .                  | Page 16 |
| Additions, corrections to manuals . . . . . | Page 17 |



## Stack and heap allocation

The stack starts at the top of the initial allocation and expands down toward low memory. A "stack pointer" points to the bottom of the active stack; as the stack grows, the stack pointer (an address) becomes smaller. The stack cannot grow below the lower limit of the initial allocation.

Demands for heap memory (the use of **new** or the opening of files) cause the task to be extended (in 1K-word chunks). The additional memory is allocated to the heap, up to the system maximum (usually 32K). When the task can be extended no farther, the heap then uses any space available at the *bottom* of the initial allocation, that is, below the stack. The heap will continue to grow upward. The stack will continue to grow downward. If the heap and stack collide, the task is "out of memory".

Note that task extension creates additional space for the heap, but not for the stack. At most, the stack can use all of the space initially allocated (either the default of 2K words, or the amount explicitly allocated at task-build time). Therefore, a program that makes heavy use of the stack (for local variables or recursion) will require explicit allocation for the **\$\$HEAP** PSECT at task-build time.

## RSX requirements for task extension

As you can see, the ability to extend the task at run-time is crucial for automatic allocation of the heap and stack. Each of the following points should be checked to allow task extension.

The **EXTK\$** directive should be included in the RSX system during SYSGEN. **EXTK\$** is used by other system software, so it is usually present; it may not, however, be available on RSX-11S systems.

For a task to be extended, it must be checkpointable (which is why the **/CP** switch should be used when in the building of Pascal tasks). Also, the checkpoint file must be active. The privileged MCR command **ACS** is used to allocate checkpoint files.

Additionally, the system manager can control the maximum amount of memory that a task is allowed to add to its original size. If programs are running out of memory before reaching the 32K-word limit, use the **SET /MAXEXT** command to determine the maximum extension allowed: the limit may be less than 32K words.

## Diagnosing errors

The error message, "Not enough memory. Make task checkpointable or extend **\$\$HEAP**", means that not enough memory is available to allocate 2K words for the Pascal

stack, or, for Pascal-1, to allocate the global-level variables. This error occurs at initialization; the program has not yet started. If you've already made the task checkpointable or extended **\$\$HEAP**, the problem is that the maximum task extension is set too low, **EXTK\$** is not enabled, or checkpointing is not enabled.

The error message "Stack overflow" means the stack is too small to hold all the local variables. A procedure containing a local array of 4500 integers, for instance, will produce a "stack overflow" even when plenty of memory is available. The array requires 4500 words on the stack, compared to the default size of 2K words (2048 words). Task-building the program with the **\$\$HEAP** PSECT extended to 5K words will solve the problem.

The error message "Not enough memory" means that requests for memory from the heap (**new** for example) exceed the available memory. If a program runs out of memory because of heap accesses, check to see whether you can close any files, thus freeing up the buffer space and control blocks they are using.

## If task extension isn't possible

If a task cannot be checkpointed, or if the extend-task directive is not available, additional space cannot be automatically allocated. In this case, you must estimate the amount of memory required for the combined heap and stack. Take into account the number of open files (at about 600 bytes per file), the size of local variables, and, for Pascal-1, space for global variables. See the section "Storage Allocation" in the Pascal-1 or Pascal-2 Programmer's Guides. For instance, a procedure with a local array of 8000 integers will require 8000 words (16,000 bytes) of stack space for the array. It's better to over-estimate (so your program will run), but avoid using too much memory, because a non-checkpointable task occupies physical memory until it finishes.

Then, explicitly allocate the estimated space for **\$\$HEAP** with the Task Builder option **EXTSCT=\$\$HEAP:num**, where **num** is the desired size of the PSECT. (**Num** must be expressed in octal bytes.)

## To complicate the issue ...

Sometimes, you may want to prevent a Pascal task from growing, yet still want it to be checkpointable. An example is a collection of Pascal tasks communicating with one other. If each task were allowed to grow to 32K words, the loading of one task could cause another task to be swapped



out. Another example involves programs making special use of virtual memory via the PLAS directives. Patching the global symbol \$NOEXT as a Task Builder option will prevent Pascal tasks from requesting more memory, yet keep them checkpointable. The format is shown in this example with a program called PROG:

```
GBLPAT=PROG:$NOEXT:240
```

Since this patch prevents task extension, you must explicitly allocate space in \$\$HEAP, as described above.

### Overlays

Even after using all the methods described above, the program still may run out of memory. The only alternative is to overlay the program.

If you must use overlays, inspect the file PAS.ODL distributed with the Pascal-1 and Pascal-2 systems for RSX. This overlay description file shows how to overlay the modules in the Pascal support library and the system library, thus saving room.

See also the "Overlays" section of the Programmer's Guide in the User's Manual.

## Changes in name, address

"Request to Amend" forms are now being sent with all new software shipments and updates.

Our license agreements require a customer to complete the "Request to Amend" form whenever the name or address of the customer changes or a new Designated Contact Person is named. The appropriate changes should be noted on the form, and an authorized official should sign the amendment.

Customers should mail the amendment to our sales department. We'll return a copy signed by one of our authorized officials. Completion of the form constitutes a legal change in the license agreement.

## Reviews

(**Introduction to Pascal**, Rodney Zaks, Sybex, Inc., Berkeley, California, 1981. 422 pages, \$14.95 softcover; no hardcover. Available through Oregon Software and elsewhere.)

Whether you're new to programming or simply new to Pascal, this book is without question the best organized and the most clearly written of the many introductory Pascal books now available.

**Introduction to Pascal** combines clear discussions of Pascal with the basics of programming techniques — a double plus for beginners. The book's strong organization helps intermediate programmers find specific information quickly and easily. Advanced programmers will also find the book useful, though occasionally lacking in depth. (Jensen and Wirth's **User Manual and Report** is probably the next place to go.)

Well-designed examples demonstrate each topic clearly and reinforce concepts that have already been presented. Just as important, the examples avoid extraneous points. The illustrations and examples are nicely set off from the text.

The order in which topics are presented is ideal, especially for beginners: as much as possible, subjects are not used or referenced before being fully described.

Sets and pointers are usually the most difficult data structures to handle in textbooks. Zaks' approach is better than average.

Zaks describes sets well, but he fails to adequately describe why a programmer would choose sets over other data structures in a particular situation. Zaks also describes pointers well, using list structures as examples, but again fails to explain the broad range of uses for pointers. Some pointer examples are somewhat large.

The description of variant records is weak and does not include any examples of usage (just a simple example of declaration). Advanced examples are missing.

A few minor errors occur, some caused by recent shifting of the ISO draft standard. The answer section does not always cover enough of the exercises.

Overall, however, **Introduction to Pascal** is clearly organized, well written, and has complete coverage of the Pascal language. The book is excellent for beginning and intermediate programmers.

(This book is also reviewed by Arthur Sale in the April 1981 — but just published — issue of the Pascal Users' Group Pascal News.)

— Will Neuhauser



## ***Pascal-2 announced for RSTS, RT-11***

Pascal-2 has now been released to RSTS/E and RT-11 customers.

The high-performance optimizing compiler is available on RSTS/E V6C and V7 and on RT-11 V3, V4, SJ, XM, and TSX-Plus. Pascal-2 was earlier released for RSX, RSX-11M, RSX-11M Plus, IAS, and VAX/VMS (compatibility mode).

Pascal-2 is a transportable five-pass compiler that emphasizes conformance to the Pascal standard while generating optimized object code. Written in Pascal, Pascal-2 will allow programs to be transported between computer systems with few changes.

The Pascal-2 compiler is larger and compiles more slowly than the Pascal-1 compiler, but produces code that is shorter and faster. Typical programs are 30 to 40 percent smaller and twice as fast. Error reporting is significantly improved over Pascal-1.

### **Discount Offered**

Customers holding Pascal-1 licenses as of August 31, 1981, may claim a discount equal to 100 percent of the current Pascal-1 license price toward the purchase of an equivalent Pascal-2 license. For instance, the current price for a Pascal-1 U.S. site license is \$1950. The same license for Pascal-2 is \$3950. A Pascal-1 user can therefore purchase the Pascal-2 license for \$2000. International customers will receive a similar discount toward their purchases, based upon international prices.

This discount will apply to all Pascal-2 orders received by December 31, 1981.

Design of Pascal-2 began in 1975. The first version of the compiler was completed in 1977. To include better ways of doing things, Pascal-2 has since been completely rewritten twice.

The Pascal-2 system consists of the compiler and the Debugger, in binary form; a number of utility programs, in source form; and the documentation, delivered in both printed and machine-readable form. The source versions of the compiler, Debugger, and run-time support library are available at extra cost, with a special license arrangements.

The Pascal-2 system has these major features:

- Code is optimized through constant folding, expression targeting, common tail merging, common subexpression elimination, and other techniques.

- Packed structures are implemented, saving space in the representation of data;
- A special "include" lexical directive allows the inclusion of multiple source files;
- A "structured constants" feature allows the initialization of data at compile time rather than at run time, thus saving space and making programming easier;
- Several low-level language extensions are implemented, including direct calls to assembly language or FORTRAN subroutines and direct memory addressing.
- Pascal-2 will give the source location for nearly 150 types of syntax errors and will detect many more run-time errors than Pascal-1.
- Strict interpretation of the emerging Pascal standard eases conversion of programs to other systems. Pascal-2 extensions can be flagged to indicate non-standard language usage.

The Pascal-2 system includes a high-level Debugger that helps programmers uncover errors that cannot be caught at compilation time. Pascal-2 also includes a collection of utility programs, written in Pascal.

The user manual is roughly twice the size of the Pascal-1 manual and contains many more examples.

Pascal-2, because it uses the same run-time support package as Pascal-1, is the logical next step for Pascal-1 users who want higher performance. Pascal-1 users, however, will continue to receive support and can look forward to further improvements to that product.

Newsletter No. 2 describes the Pascal-2 system in detail and lists the major differences between the implementation of Pascal-2 and of Pascal-1. These differences are also catalogued in the Conversion Guide, included in the Pascal-2 manual, which explains ways to convert Pascal-1 programs to Pascal-2 programs.

### **Note for RT-11 users**

A single-density floppy (RX01) functioning as the system device is not large enough to run Pascal-2. An RX01 disk can be used as the release mechanism if the system device is double-density (RX02) or larger. Many users with RX01-based systems have found the step up to an RX02-based system well worth the modest cost. The only other RT-11 restriction is that the Pascal-2 text formatter, Prose, is too large to run under the XM monitor; Prose must run under the SJ monitor.

## Letters

**Editor's Note:** Anyone who writes us expecting a technical reply can also expect the question to be published in this newsletter. (If you need an answer, others probably do too.) We will condense letters and correct grammar and spelling as needed. We will honor a request not to publish a letter, but we may generalize the question and run the letter anonymously.

### To the editor:

We have been using the Pascal-1 compiler to teach Pascal for the last year, and I thought you might be interested in hearing our thoughts on the system.

First the details of our system and the compiler. The machine is a PDP-11/45 with 456K bytes of memory running under Version 7.0 of RSTS/E. We have been using Version 1.2E of the compiler. The compiler was used by 160 different students in two introductory Pascal courses.

Overall, we are reasonably happy with the performance of the compiler. It was reliable and performed efficiently considering the heavy work it was asked to undertake. We would like to continue to use the system in the future, but we are unlikely to move to Pascal-2. The reason for this is that "Pascal-2 is larger and compiles more slowly than Pascal-1" (Oregon Software Newsletter No. 2). Student programs tend to contain compilation errors, and even when they successfully compile they are unlikely to execute for very long. Thus the emphasis in Pascal-2 is the exact opposite of what we require in an educational establishment.

The following is a list of bugs or comments about the system:

(1) Would it be possible to extend the range of **set of char**? The lack of lower-case letters in sets is most regrettable.

(2) Run-time errors are reported with an indication of the program counter. From this there is no way of knowing where the error is located in the source. One way of overcoming this would be to have an extra column in the compilation listing containing the number of instructions generated up to the start of that line. The run-time system could also be made aware of the load point and by a simple subtraction could give the location of the error relative to the start of the program. The user could then discover from his compilation listing the line on which the error occurred.

(3) One of the most powerful features of Pascal is the sub-range facility. This enables the development of reliable and readable programs. By not checking assignments to sub-range variables either at compile time or run time the only argument for using subranges is for readability. Similarly no run-time checks are made on input to subrange variables. Checks on subrange variables are extremely important; this is the worst deficiency of your system.

(4) The correct response to a case expression being out of bounds is to give a run-time error, not to skip over the case.

(5) Integer overflow also needs to be checked. If you feel that all these run-time checks are expensive, they could be made an option, but by default switched on (the "putting to sea with lifejackets on" theory).

(6) The following program executes:

```
program pointers(output);
type
  arec = record i,j :integer end;
  aptr = ^arec;
var
  ptr :aptr;

begin
  new(ptr); dispose(ptr);
  ptr^.i :=27
end.
```

Why not check every pointer access to see whether it points inside the heap? If **dispose** changed the value of **ptr** to something outside the heap, then this sort of error could be detected.

Although the above might seem critical of your system, my main aim is to persuade you to develop your product from the current status of good to the status of excellent.

Dr. D. J. Robson  
Computer Studies Group  
University of Southampton  
United Kingdom

### Editor's Reply:

Perhaps our emphasis on performance of the code generated by Pascal-2 has left the impression that raw performance is the only new feature. Not so! Pascal-2 was designed

Continued on Page 6



with educational environments in mind, and with the knowledge that compilation speed, error checking, and run-time safety can be very important.

Thus, Pascal-2 checks for errors in the first two passes. If any are found, the compiler skips successive steps, including optimization. As a result, an incorrect program will compile nearly as fast with Pascal-2 as with Pascal-1.

More important is the problem of error checking in student programs. Pascal-2 detects many more syntactical errors and gives much more meaningful error messages than does Pascal-1. In addition, Pascal-2 detects a number of run-time errors that Pascal-1 does not. Pascal-2 is therefore much more likely to get beginners beyond syntactical blunders and common run-time errors and on to writing functional programs.

Other advantages of Pascal-2 over Pascal-1 are illustrated by your questions about limitations of Pascal-1:

Pascal-2 has 256-element sets, and Pascal-1 does not. Pascal-2 checks subrange assignments. Pascal-2 gives an error message when a case expression is out of bounds. Pascal-2 checks integer overflow for multiply and divide, which are the operations most likely to create an overflow.

Neither Pascal-1 nor Pascal-2 presently detects the pointer error on the PDP-11. However, upon inspecting the above example, we realized that Pascal-2 can be made to flag that particular misuse, at compile time. (It's an example of the undefined variable checking performed by the compiler.) We're working on that fix now.

The compile-time checking can find many, but not all, of these dynamic errors. Ideally, as you suggest, run-time checking should be available for all possible errors. However, the hardware design of the PDP-11 requires substantial overhead for some kinds of run-time error-checking, notably undefined variables and integer overflow. Other hardware designs, such as the VAX, permit more error checking at reasonable cost.

Your suggestions for changes in the listing to help detect run-time errors are valid. However, the interactive Debugger supplied with both Pascal-1 and Pascal-2 accomplishes the same thing. The Debugger automatically halts when it detects an error in the program and shows the source program statement at which the error occurs. Other Debugger facilities can then help the user determine the cause of the error.

John Anderson of the Zoology Dept. of the University of Washington in Seattle (USA) asks these questions after reading the Pascal-2 manual for RSX:

**Q.** I thought **forward** is standard, but Page 64 shows **forward** as a Pascal-2 extension. However, Page 70 shows no asterisk before **forward**, indicating that it is standard. Which is it?

**A.** **Forward** is a standard directive under the ISO standard. The problem is that the standard treats a directive as neither an identifier nor a reserved word. As described on Page 61 of the RSX Language Specification, we have chosen to treat **forward** as a reserved word. Therefore, our documentation is correct: **forward** is both a standard directive (Page 70) and a non-standard reserved word (Page 64).

**Q.** Also on Page 64, **emt** is listed as a predefined procedure, but it is not discussed anywhere. Shouldn't **emt** be explained in the section on low-level interfaces?

**A.** The **emt** ("emulator trap") procedure applies only to the RSTS/E operating system. The RSX and RT-11 manuals should have contained notes to this effect. The RSTS manual contains a brief definition of **emt** in the Language Specification; however, a more detailed **emt** section was inadvertently left out of the RSTS manual Programmer's Guide and is being sent to RSTS users as an addendum.

**Q.** The discussion (Page 55) of direct access does not make clear whether **/seek** is limited to 1 record/block. If **/seek** is limited to 1 record/block, what happens to long records?

**A.** The **/seek** switch refers to components in a file, not blocks on a disk. **/Seek** is **not** limited to 1 record/block, and records longer than a block are also supported. (In general, we've tried to document our Pascal products so that if a restriction is not mentioned, you may assume that unrestricted use is supported.)

**Q.** What happens with undetected run-time errors, such as **abs(-32768)**?

**A.** The most common cases involve integer overflow and uninitialized variables that escape detection by the compiler.

Signed integer overflow generally results in a value with a sign opposite from the mathematically expected result. For instance, the expression **abs(-32768)** yields -32768, not 32768, whereas the expression **maxint+2** yields -32767, rather than 32769. Unsigned integer overflow generally results in a value modulo 65536. The above is true for

addition and subtraction; results for multiply and divide depend upon which model PDP-11 processor is in use.

Uninitialized variables can cause unpredictable results, depending upon type. Pointers will often cause processor traps, since the value might be odd or might point outside the legal address space provided by the operating system. An uninitialized variable other than a pointer may cause unpredictable results, with the program "working" once and failing a second time, or failing differently each time it is executed.

The Xref utility can help uncover uninitialized variables. This cross-referencer marks each assignment to a variable; it is a good practice to check that each variable in a program is assigned a value at least once. This does not guarantee that the variable is initialized before use, but the approach can uncover some simple cases undetected at compile time.

## 'For' loop restriction

A **for** loop with an index variable that has been declared with an extended range will loop indefinitely upon occasion. The following example demonstrates the failing cases:

```
var i:0..65535; {an extended-range variable}

begin
  { Any final value less than 65535 works
    correctly, but the loop never terminates. }

  for i:=0 to 65535 do writeln('never halts');

  { Corresponding case for downto loops }

  for i:=10 downto 0 do writeln('never halts');

end.
```

The problem stems from the fact that the PDP-11 INC and DEC instructions do not set the carry condition code upon unsigned overflow.

This **for** loop restriction may be removed in a future release.

## Information Exchange

If you need information on technical applications involving Pascal, or if you have an application that might be of interest to other users, send us a brief description for inclusion in the Information Exchange. Interested parties can contact one another directly. Follow the format of the items below.

**Pascal Users Group** for Pascal-1 and Pascal-2 is being formed to share the methodology in solving tasks with Pascal-1 and Pascal-2 and to spread information on problems and solutions with those products. Bruce Williams, Pascal User's Group, EOCOM, 15771 Redhill Avenue, Tustin, California 92680.

**Portable database system** (RSX, RT-11, Honeywell); 10 systems in the field. Dr. B. Gliss, c/o MPI Stuttgart, Heisenbergstr. 1, D-7000 Stuttgart 80, Germany. Tel. 0(711) 7830-251.

**Q.** We are using Pascal-1 (V1.2) on a PDP-11/23 with the RT-11 operating system and want to do interactive graphics. Is anybody using FORTRAN graphics subroutines in Pascal programs? Is anybody working on a Pascal graphics package? We also need device drivers for Tektronix equipment (4010, 4662). Gini Van Siclen (301-338-8344) or Dave Elliott (301-338-7049), Dept. of Earth and Planetary Science, Johns Hopkins University, Baltimore, MD 21218.

**Menu-driven modeling** and reporting systems designed for use by financial and planning oriented individuals; more than 200 routines in design and data manipulation, reporting and file handling; for PDP-11 (RSTS/E), VAX (VMS), Hewlett Packard 3000 (MPE). Canadian European Systems Ltd, P.O. Box 2884, Vancouver, B.C. Canada, V6B 3X4, (604) 732-9141.

**Networking program** that uses a master/slave distribution of processing; synchronous processing of messages received on multiple asynchronous ports; more than 90% in Pascal-1; EOCOM, a division of American Hoechst Corp., 15771 Redhill Avenue, Tustin, CA 92680.



## Trouble with user trouble reports

A cautionary tale: On Friday, February 13, 1981, Joseph R. Smithson of Harnsford-on-Flushing in Middlesex, England, posted a trouble report about his Pascal-1 compiler to Oregon Software in Portland, Oregon, USA. In late March, Mr. Smithson began wondering about Oregon Software's claim of 30-day turnaround of trouble reports.

On Wednesday, April 1, Mr. Smithson sent a rather nasty wire to Oregon Software about the tardy response. And on Thursday, April 2, Mr. Smithson's original letter arrived at Oregon Software — via surface mail.

Yes, it really happens. We're less eager to admit that late responses are sometimes the result of our mistakes.

We want **all** Trouble Reports to receive a response within 30 days. Toward that goal we log every report and monitor its progress through our system. And even though our customer base has quadrupled in the past two years, we usually send an answer within 30 days. If you feel we've missed you, please call us collect or send us a Telex.

Steve Poulsen, a key member of our programming group, tells us that the main cause of delay in our response is the lack of sufficient documentation to permit us to reproduce the trouble. Will Neuhauser, who does preliminary screening and logging of each Trouble Report, tells us that slightly more than one-third of the reports are returned to the customer for additional information.

Overseas customers must use air mail or air express. In a rare case, a sample program may be short enough to Telex. But remember, our 30-day response guarantee starts when we receive the fully documented Trouble Report.

Other things you can do to help us serve you better include:

- Use our Trouble Report; send the reports to our sales department, which will coordinate our response. If you don't have a Trouble Report form, write anyway. Be sure to include your site number (e.g., #1-334); compiler version number (e.g., 2.0I); operating system version (e.g., RSX-11M-Plus V3.2), and a description that is sufficient for us to reproduce the bug or understand the problem. (And have us send you a pile of Trouble Reports.)

- Send the **shortest** sample program that shows the problem. Programs that are too long to enter onto our system by hand will be returned with a request that it be resubmitted in a machine-readable form.

A short example program demonstrating the problem can be evaluated much more quickly than a long program with "include" files and external functions and procedures. Interactive programs are especially difficult to analyze, so

### Distributor customers

If you are a distributor customer, direct all Trouble Reports to your distributor instead of to Oregon Software. Our distributors have technical support staff to assist you, and our distributors will contact us in case of particularly prickly technical problems. Your distributor needs the same information that we do to process trouble reports: Site number, compiler version, operating system and version number, detailed descriptions in readable format.

give sample input and be sure that the program provides prompts for all required input.

- If the program is more than a page, send it on an RT format floppy or a 9-track DOS format magtape. Place a written label on the media, with your site number, the format of the media (DOS, ASCII, FLX, RT-11), the density (single-density or double-density floppies, 800 or 1600 bpi tapes), the label used to identify the media to the system (if applicable) and the directory used to identify the media to the system. Don't send us confidential materials.

- If the bug is "fixed in next release", you can speed getting your free copy of the next version by having your Designated Contact Person sign the request on the Trouble Report before you send it.

### Telephone requests

Before you call, ask yourself these two questions: Are you in support? Are you the Designated Contact Person? If the answer to both questions is yes, then call and ask for Technical Support. Be sure to have ready those magic numbers: site number and compiler version number (both at the top of any listing), and the operating system and version number.

We will try to determine the nature of the problem and give any quick fixes we know.

If, as a result of the call, you want an update to Pascal-1 or Pascal-2, the Designated Contact Person must request that update in writing. For legal reasons, we need a written request for each update, even though the update is free if you are in support.

If you aren't satisfied with your service, then call, write, or Telex directly to our president, Rusty Whitney.

# Text formatting with Pascal-1

By DAVE THOMAS

(Mr. Thomas is with Novus Systems Technology Ltd., Koninginnegracht 56, 2514 AE, Den Haag, The Netherlands. In addition to the text-formatting system, Systems Technology has also used Pascal to develop a test and monitoring system for telex and telephone lines and to develop a telephone directory system.)

This article describes the use of Pascal for text formatting, specifically a set of programs collectively known as IGOR. This article may be of interest in two ways. First, some features provided in IGOR are normally found only on formatters running on large machines. Second, some implementation techniques described (in particular the support of variable-length strings in Pascal-1) may be of use to some readers.

## Background

As a small software company, Systems Technology found itself spending a lot of time producing documentation. Typically in the course of a project we would write letters, tenders, contracts, various specifications, a user manual and a programmer's guide.

We looked for a text-formatting package available on our machines (LSI/11-23s with RX02 floppy diskettes or RL01 hard disks). However, we found none that met our primary requirements: document-level formatting commands; tailorability to our formats; support for simple graphics; support for various printers.

The first point was particularly important. The formatter was to be used by both technical and secretarial staff. Neither group would be willing to learn a complex series of commands to perform conceptually simple operations, such as starting a chapter or adding an element to a list of numbered points. As well as simplifying document preparation, this "high-level" approach ensures that all documents conform to a house style.

Given this apparent vacuum, we decided to implement our own formatter. (Besides, it was more fun.)

## The IGOR Solution

We discarded the "word processing" approach ("what you see is what you get") because it entails losing too much information about the document structure. (Kernighan summarizes this loss as "what you see is all you've got").

Instead we decided to go for an off-line text formatter, much in the style of Script on the IBM 370 series or RUNOFF on DEC machines. An off-line text formatter processes the document, which contains formatting directives, to produce the final (printable) copy. The entire document is reprocessed after any change.

The IGOR system consists of a formatting program and a group of output processors, one for each device type. The formatting program takes a **source document** and a **document profile**, producing a formatted intermediate file. This intermediate file is run through the output processor appropriate to the device being used. The same intermediate file may be used many times, so that draft and final copy may be produced without reformatting the document.

Because most of our technical people enjoy a heritage of Script-like systems, we adopted the Script (and RUNOFF) philosophy for the source-document files. The document contains a mixture of text and formatting directives. An extract from a typical IGOR document is:

```
There are several objections to this
algorithm.
.sp;.ce ** more needed **
.Section 'Storage constraints'
On the target machine . . .
```

The text in the document is formatted according to the various format options. By default, text is word-wrapped between lines and justified to two margins. Commands are indicated by a special character at the start of a logical line. In the example above, the two commands **.sp** and **.ce \*\* more needed \*\*** tell IGOR to space down one line and center the text (which is a note to the reader that part of the text is missing). The command **.Section** starts a new major section in the document with the given title. Text for the section then follows.

A feature possibly unique to IGOR is a simple means of specifying line graphics. Technical documentation often contains diagrams and tables using only horizontal and vertical lines, with the appropriate corners. IGOR allows these lines to be specified directly in the text of the document by means of a **graphics character**. The diagram is typed into the document via this graphics character wherever a line segment is needed. On output, IGOR looks at the neighbors of each graphics character to determine whether the character represents a line segment, an intersection or a corner. IGOR then does its best to represent the resulting graphic on the output device.

Continued on Page 10



As an example, a writer would type:

An **imbed** facility allows a document to be constructed from more than one source. Most of our manuals are split into chapters, one per computer file. This allows individual chapters to be formatted for proof-reading. Standard paragraphs in contracts and tenders may be imbedded in the same way.

Before the document body is processed, the IGOR formatter automatically reads the appropriate **document profile**. A separate profile is written for each document type (report, contract, etc.) The profile serves two main purposes. First, it establishes document characteristics such as page length and heading style. Second, it defines a set of document-specific commands. For example, the profile for reports will define commands for starting chapters and sections; the profile for letters will provide commands for generating greetings and farewells. New document types may be defined with appropriate profiles.

The intermediate file contains the formatted document in a device-independent form. The output processor for a particular device will translate this into a sequence of characters suitable for that device. One of the most important translations is the handling of line graphics. The serial printers we use for draft output do not support graphics. The output processor therefore translates the graphic sequences into available characters such as hyphens, vertical bars and plus signs. A rough version of document graphics may therefore be produced even on non-graphic devices.

The output processors are all screen-based. The document may be viewed, one page at a time, as it is being printed. Pages may be printed selectively or automatically.

### Easily Extended

IGOR has features that make it easy to use and extend.

**Variables:** Arbitrary strings or numbers may be assigned to named variables. These may be manipulated and substituted into the document body. A common use of variables is in contracts. The name of the client is assigned to a variable at the start of the document. The standard paragraphs that follow may use this variable when they need to refer to the client and therefore need not be changed when a contract is drawn up for a different client.

**Interaction:** Text may be written to and read from the terminal during formatting. This allows data to be prompted for and included in the document, as well as providing a means of controlling the formatting itself.

**Arithmetic:** IGOR supports simple four-function integer arithmetic. As well as the more mundane uses, such as allowing chapters and sections to be numbered automati-

cally, the arithmetic capabilities allow IGOR to be used as a simple calculator. Document profiles have been written to keep track of expenses and timesheets, with automatic calculation of purchase tax, subtotals and totals.

**Conditional sections:** Sections of the document may be conditionally excluded from the formatting process. One use of this is to produce two versions of the same document (one perhaps containing confidential material).

**Additional commands:** Formatting directives and text may be packaged together to form a **remote**. Remotes and IGOR commands are invoked identically, so that the user need not be aware of the distinction. When remotes are defined in the document profile, additional document-specific commands become available to the user. The **Chapter** command used in reports is actually a remote. It may be modified at each site to customize it to that site's requirements.

(Remotes may invoke other remotes, including themselves. This recursive property, combined with the conditional facility and variables, makes IGOR a usable programming language. There even exists a version of Towers of Hanoi written in IGOR.)

Remotes may be defined by writers of documents. Indeed, writers who are programmers often will expend more effort on the remotes than on the document itself. Occasionally a remote turns out to have general application, in which case we add it to a library. One such remote creates complex file description tables, automatically keeping track of field offsets within records. It probably took the programmer three or four days to get this remote right, but it has since saved us a lot of time and trouble. This facility also means that all the record descriptions in all our documents look the same, no matter who wrote them.

### Implementation

The implementation of IGOR gave us the opportunity to learn a lot about using Pascal-1. We'll look at two aspects of the implementation in detail. The first is a "dirty trick" used right in the heart of IGOR to provide variable-length strings. Although the use is not something to be proud of, we feel that it may help other readers who have similar requirements.

The second aspect of the implementation is the management of the program source. We discuss both the compilation technique used and a method we used for verifying the correctness of the overlay structure chosen.

Continued on Page 11

## Variable-Length Strings

Most of the data structures used in IGOR are lists or trees. However, it seemed more efficient to hold the values of variables and the bodies of remotes as arrays of characters. Because of space constraints, we needed to keep the storage used by each string to a minimum. The two normal solutions to this problem are linked lists of fixed-length arrays or allocation from a large static array. The first was too expensive in space. The second put an artificial limit on the amount of space available for variable storage. We therefore decided to implement true variable-length strings.

The solution we adopted is specific to Pascal-1, although the routines as developed could probably be linked into Pascal-2 programs with no ill effects. The general principle will work with any Pascal system that performs true run-time heap management.

A flexible string is defined to be a pointer to a record:

```
type
  FlexibleString = ^FlexibleStringRecord;

FlexibleStringRecord
  = record
    Length  : FlexLength;
    Chars   : array [FlexLength] of char
  end;
```

where **FlexLength** is some suitably large subrange (we use 1..9999).

In the same way that normal dynamic heap items are created and deleted by **new** and **dispose**, we define two procedures **FNew(String, Length)** and **FDiscard(String)** to handle flexible strings. The **FNew** procedure creates a flexible string record of the given length. The characters in the string are accessed as **String^.Chars[i]**.

Internally, **FNew** and **FDiscard** use imbedded MACRO to access the Pascal run-time routines \$B70 and \$B72. \$B70 is the routine called whenever a **new** is executed. It is passed the length required (rounded to a full word) and returns the address of a suitable free area on the heap. Both values are passed on the top of the stack. \$B72 handles **dispose**. It is passed the size to free in register 0 and the address of the area to free on the top of stack.

To complicate things slightly further, we actually create the string 4 bytes longer than requested. Some of this extra length gets round a bug with the earlier versions of Pascal-1 that caused \$B72 occasionally to free too much store. One additional byte is used to hold a sentinel. This is initialized by the **FNew** procedure. When **FDiscard** is called, it checks for the presence of the sentinel value. If

absent, it means that something attempted to write beyond the end of the string, and that program integrity has been lost.

The source of the two procedures is given on the accompanying page. It is probably unlikely that Oregon Software will support programs that use \$B70 and \$B72 in this way.

## Managing the source

IGOR is implemented almost entirely in Pascal-1. The source (approximately 15,000 lines) is spread over about 80 RT-11 files. This makes the average file less than 200 lines long, resulting in easy editing and fast compilations. Extensive use is made of the external compilation feature of Pascal-1.

Each major procedure or function in IGOR is held in a separate source file and is compiled to produce a separate object file. The global constants, types and variables are held in three files. The PCL program supplied with Pascal-1 was modified to prefix these globals to the file being compiled. Thus to compile a component of the formatter, type:

```
.RUN IPCL <component name>
```

As space is a problem at run-time, the optimizer IMP is used when generating a production (as opposed to test) version of the IGOR formatter. The overall improvement in space was found to be about 6.8%.

The IGOR formatter is heavily overlaid (as the non-overlaid program size is about three times the size of the PDP-11's address space). The administration of a program with some 40 overlaid procedures, some recursive, is not easy! To ensure that the return path is never destroyed, we eventually resorted to drawing a 40-by-40 matrix of "what calls what". A quick program to find the transitive closure of this at least allows us to check that a particular overlay structure is valid.

(An aside to the interested reader with time to spare. A utility program that produces a list of procedures imported and exported by a Pascal program segment would be most welcome. The information needed seems to be in the symbol table file output by the compiler.)

Continued on Page 12



```

procedure FNew(var String : FlexibleString;
               Size : FlexLength);
var
  LocalString : FlexibleString;
  LocalSize   : PositiveInteger;
begin
  (* Round up size and allow for sentinel *)
  LocalSize := Size + ord (odd (Size)) + 4;

  (*$C .GLOBL $B70
  MOV   LocalSize(6),%0      ; stack size to get
  MOV   %0,-(6)              ; on any m/c
  JSR   %7,$B70              ; call NEW
  MOV   (6)+,%0              ; pop address
  MOV   %0,LocalString(6)    ; and save it
  *)
  (* Set length and sentinel *)

  with LocalString^ do begin
    Length := LocalSize;
    Chars[Length-2] := Sentinel;
  end;

  String := LocalString
end ;

procedure FDispose(var String : FlexibleString);
var
  LocalString : FlexibleString;
  LocalSize   : PositiveInteger;
begin
  LocalString := String;

  if String = nil
  then <error action>
  else begin
    LocalSize := String^.Length;
    if String^.Chars[LocalSize-2] <> Sentinel
    then <error action>
    else begin
      (*$C
      .GLOBL $B72
      MOV   LocalSize(6),%0      ; size to free
      MOV   LocalString(6),%1    ; stack address
      MOV   %1,-(6)              ; on any m/c
      JSR   %7,$B72              ; call DISPOSE
      *)
      String := nil ;
    end;
  end;
end;
end;

```

## Conclusions

We were very pleased with the performance and reliability of Pascal-1 throughout the implementation of the IGOR system. The external compilation system, the ability to embed MACRO and the control possible with **reset** and **rewrite** were particularly appreciated.

However, lest success go to their heads, we do have several moans, none serious. We would have liked to use **set of char** several times, but were forced to code around it by the 64-element limit. Similarly, **packed** would save quite a bit of space at run time. Both of these features are included in the new Pascal-2 compiler.

The only other real complaint is user error handling. The run-time error procedure is passed an integer error code (good), but that code is always zero (not so good). It is useful to know what the error is before knowing whether the program can safely ignore it.

Despite these minor grouses, we are very happy with both Pascal-1 and the text formatter we implemented with it.

**Editor's Note:** This particular use of MACRO to access the Pascal run-time routines \$B70 and \$B72 is not unreasonable, but we caution any user about using embedded MACRO commands that depend upon our support library routines remaining constant forever. We cannot guarantee that will be true. (In particular, the internal calling sequences for new and dispose will be changed.)

Pascal-1 programs containing embedded assembly code can be linked to Pascal-2 programs with "no ill effect". See the Conversion Guide of the Pascal-2 User's Manual.

Systems Technology had to work around two problems that no longer exist. We've fixed the bug that required the variable string to be 4 bytes longer than needed. Also, beginning with V1.2H (and 2.0H), the error procedure code returns an integer value corresponding to the particular run-time error. Even so, the user's possible courses of action are still limited. See the Programmer's Guide of your User Manual for details.

As for Mr. Thomas's desire for a procedure cross-referencer, Pascal-2 has such an animal. As he notes, Pascal-2 also implements 256-element sets and packed structures.

# The Log: Pascal-1 and Pascal-2

Oregon Software's first two Pascal Newsletters described the significant changes in Pascal-1 V1.2 from its initial release, V1.2A, in December 1979 through V1.2H in June 1981. This issue describes the significant changes in V1.2 through its present release, V1.2J.

This log also describes the changes in Pascal-2 since its initial release, V2.0H, in June 1981 through its present release, V2.0J.

To use this log, first determine what release of V1.2 or V2.0 you have. (The release number is printed on the headline of all program listings.) Then review the log to determine the changes made since the release of your version. If the changes are of particular importance to your application, the Designated Contact Person should request an update, in writing. The DCP is usually the senior programmer in charge of software. You will receive the latest version of the software.

The first section of this log describes changes applicable to Pascal-1 for all operating systems. Following sections describe Pascal-1 changes specific to each operating system. The version listed is the one in which the change first appears; succeeding versions also include the change. The same format follows for Pascal-2 (most Pascal-2 changes were for all systems, however.)

Future issues of the newsletter will continue to outline improvements to Oregon Software's Pascal products.

## Pascal-1

V1.2J corrected these problems:

### Incorrect code for some calculations

Compiler generated incorrect code for some mixed **real/integer** comparisons (operands were sometimes being reversed on the stack). Compiler also was not marking the result register of an **abs()** calculation as used, causing the register contents to be destroyed by later calculations.

### Procedure parameter in nested scopes

Calling a procedure parameter from an inner scope didn't work.

### Illegal scalar redefinition

Illegal redefinition of a scalar identifier as in (**red,white,blue**) as a procedure identifier (**procedure white**) would crash compiler with an odd address trap if the ordinal of the scalar was odd.

## Changes to RSTS/E

None.

## Changes to RT-11

V1.2I corrected these problems:

### Unpredictable trap if file could not be opened

If the **USR** was swapped in when an error occurred, the default file channel blocks were not available. The **Error** procedure attempted to write to the default output file and produced a recursive error if the **USR** was in memory.

### 'Dispose of Nil' after reset/rewrite errors

In some cases when a program was unable to open a file, the additional error "Dispose of Nil" was generated while the program was being terminated. The error was caused by an attempt to release a buffer that had not been allocated.

### 'MMU Fault' on XM virtual jobs

If the first block allocated in the heap was later disposed of by an XM virtual job, a memory segmentation fault occurs, because a zone of unmapped memory exists between the program code and the heap. Linking with the **/U:20000** switch will eliminate the problem in earlier versions.

## Changes to RSX

V1.2J corrected these problems:

### Extra 'readln' allowed

In some cases involving **text** files, an extra **readln** was being permitted at the end of the file without signaling a "Reading past end-of-file" error. The **eof** flag was properly being set, but the file pointer was not. An extra **readln** at **eof** now correctly gives an error.



### Error with /APD switch

In some cases involving **text** files, the use of the append switch /APD would cause the support library to access an uninitialized variable. Users would see this problem as a "record length error". Depending on the uninitialized value being used, random memory may have been used as the line buffer, and this could cause traps and other random failures in the program. This problem happened because FCS rather than Pascal did the original positioning of the file.

### Output errors with write

If more than 132 bytes were available in a file's block buffer, and if the last output operation was a **write** rather than a **writeln**, the end-of-file byte count in the file's header would not be properly updated; if the file were later opened, the data in the last line was not available to FCS. **Text** output files have been changed to always use move mode rather than locate mode.

## Pascal-2

V2.0I fixed these problems:

### Packed structures

Certain types of packed structures were not being unpacked properly; record packing was improved.

### Structured constants

A structured constant containing variant records was not properly handled; a bug involving quoted string constants in constant structures was fixed.

### Integer 'not' incorrect for constants

Folding problem with **not** operator was fixed. All constant **not** operations were being done as booleans ("not 0" became 1 instead of -1).

### 'In' operator erred

**In** operator incorrectly popped part of the set operand at times, leading to unpredictable fatal errors.

### Integer 'and' gave no value

No value was generated for an integer **and** used in the context of an **if**, even when later code used the value as a common expression.

### Function results passed incorrectly

A problem was fixed in the passing of a function result as a parameter. Problem occurred when two calls were nested and when the outer call returned a value shorter than the interior call (e.g., `writeln(trunc(realfunction))` ).

### Procedure parameters passed incorrectly

A problem was fixed in the passing of procedure parameters to inner procedures.

### Functions as parameters

Error with function parameters resulted in functions erratically appearing to be **nonpascal** functions.

### 'For' statement code corrected

Three bugs were fixed in **for** statement code: **for** loops with a byte-sized control variable could give incorrect results; a **for** loop could terminate incorrectly; a **for** loop control variable was marked as undefined even if the **for** loop was contained in a conditional statement so that the **for** loop might not be executed.

### Variable references erred

Intermediate variables were not being properly referenced after a non-local **goto** or a Pascal-1 call.

### 'Out of memory' with many real constants

An inconsistency in the handling of real constants was removed. The problem could cause "out of memory" errors in the compilation of programs using many real constants.

### Reading of reals failed on non-FPP

Programs reading real numbers failed when compiled under /sim, /eis, or /fis, and a read of real numbers from a text file did not always store the result properly.

### Large exponents caused traps

Large floating exponents could cause traps at compile time, and small numbers with many digits could be diagnosed as an exponent out of range.

### Error reporting improved

A missing semicolon between parameter declarations is now detected; constant division by "0.0" now produces an error message; when an identifier is expected but not found in a type definition, user will correctly get "undefined identifier" message (previously, a "double definition" error would occur in unpredictable places).

### Listings improved

Listings were corrected to properly handle comment nesting, tabs, and page ejects.

### Reset/rewrite altered for RSTS/RT

**Reset/rewrite** calculations of file element length were changed to accommodate the way RSTS/E and RT-11 implement block files.

V2.0J fixed these problems:

### Structured constant errors caused compiler loops

Error diagnostics in structured constants were not properly handled, causing the compiler to loop or crash. For instance, an undefined identifier in a structured constant would cause the compiler to loop.

### Structured constants packed incorrectly

Structured constants were not being handled properly when an unpacked structure was included as a component of a packed structure or vice versa. This also caused Debugger problems. Code was added to make appropriate transfers when changing from one to another.

### Duplicate definition caused compiler traps

Duplicate definition of an identifier caused an odd address trap.

### Bad parameter specification caused compiler crash

If a user specified a parameter type by writing it out instead of using a type identifier, the compiler could crash before generating the proper error message.

### Spooky errors with real constants

If a constant happened to have the same binary value as an instruction that could be deleted, the constant could be deleted as an "optimization", producing strange errors.

### Bad code for integer expressions

Incorrect code was generated for some integer expressions that used both multiplication and division.

### 'In' operator error

If the left-hand operand was constant and right-hand operand was both stored in a register and 9 to 16 bits long, then the generated code failed.

### Errors with set expressions

Certain complicated set expressions would cause errors.

### More 'for' loop problems

Loop variables were inaccessible from external procedures, and some **for** loops failed to terminate properly.

### Problem with 'nonpascal' real functions

A **real** result from a **nonpascal** function was mistakenly assumed to be returned in the floating-point accumulators. All **nonpascal** function results are now taken from the general registers.

### External functions

An external function was not allowed as an actual parameter.

### Loophole produced errors

**Loophole** produced erratic results if the parameter to **loophole** was shorter than the result (e.g., when an integer was floated).

### Trunc and round

**Trunc** and **round** did not work on non-FPP machines.

### 'Undeleted temps' message

Internal compiler errors generate this message. In some cases, the compiler failed to delete temporary variables (usually generated during optimizations); in others, the compiler lost track of the stack.



### 'Out of memory in procedure x'

A number of internal changes were made to allow the compiler to handle more complex user programs.

### Error reporting improved ✓

A colon after **otherwise** (a common mistake) generates a specific error message rather than a series of general error messages.

### Unfixed

### Incorrect error for overly complex programs

Deeply nested records will give the message "Stack overflow. Try expanding `$$heap xxxxxx`". The message should say simply "out of memory". The problem is a limitation on total memory storage; the solution is to reduce the number of types or the number of nesting levels.

### UIC in default fails in reset/rewrite on RSX

The Pascal program cannot locate the file when a UIC is specified in the third parameter of **reset/rewrite** (the default file name). The problem is with the way FCS PARSE handles defaults. Temporary solution: don't specify UICs in the default name parameter.

## TSX and TSX-Plus

A program may run out of memory under TSX unless the user runs the TSX-supplied SETSIZ program. This program sets the size of memory required by the program into byte 56 of the .SAV file.

The installation process links the PCL program with a /STACK specification, which is not allowed under TSX. Simply remove the /STACK switch and relink PCL.

## RT-11 disk space problems

Occasionally, an RT-11 program that has worked consistently in the past will produce the error messages "attempting to read past end of file" or "not enough room for user on device", or will fail attempting a **rewrite** file operation. A related problem is the Pascal-2 compiler error message "rewrite failure: CACHE.TMP".

The problem may be caused by a lack of contiguous disk space for a file. The significant term here is "contiguous": be several thousand disk blocks may be available, with no single space that is large enough for the file.

The cure is to squeeze the disk with the SQUEEZE command. This command moves all of the existing files to the head of the disk, compressing the unused blocks into one contiguous region on the disk. The command is "SQUEEZE XXX:", where XXX: is the disk to be compressed. (The system will ask you to confirm the SQUEEZE command.)

**Warning:** Never squeeze a disk under TSX while other persons are using the disk. You can destroy their files.

A further consideration of disk space: Unlike RSTS and RSX, RT-11 files cannot be extended after they are created. An application that will accumulate data over several runs should, when first executed, create files that are large enough to contain all of the expected data.

### Linker transfer address problem ... again

The latest revision (V6.01E) of Digital's Linker still has a problem setting the transfer address of Pascal programs. (The problem is with overlaid programs that have their transfer address defined from a library.) The V3 Linker does not have this problem and may be used under V4; users who have only the V4 Linker should explicitly set the transfer address as follows:

```
.R LINK
*PROGRAM = PROGRAM,SY:PASCAL/T
Transfer Address ? $START
~C
```

### Pascal-2 too big for SJ BATCH

The RT-11 SJ monitor does not leave enough memory for a user to submit Pascal-2 compilations as BATCH jobs. (We're working on it.)

# Errors, additions to manuals

## Pascal-2 RSX manual

### I/O Control Switches, Page 20

Add this switch:

`/buff:n` (Buffersize): Pascal-2 normally allocates the minimum space required for a file buffer, which is usually 512 bytes but is dependent on device and file characteristics. More efficient I/O transfers can be performed at the cost of additional memory. The `/buff:n` switch specifies the storage (in decimal bytes) to be allocated to a file buffer.

### Overlays, Page 23

The correct line in the ODL example should be:

```
.ROOT SINGLE,SYSIO,*MAIN-*(SUB1,SUB2,SUB3)
```

### The Stack, Page 25

Second paragraph, first sentence should say: "The space allocated for the stack ...", not "the stack frame".

### Storage Allocation, Page 27

Second paragraph, first sentence under Set Types should say: "In this example, **Hotset** is allocated the same amount of space as **Colorset**", not "the same space".

### Structured constants, Page 54

The structured constants example in the Language Specification is incorrect. Parentheses are needed around each record in the outer array. The following is the correct text of the example.

```
type
  compensation = (paid, unpaid);

paytype = record
  title : (clerk,indian,chief,president);
  case compensation of
    paid : ( rate : real );
    unpaid: ();
  end;
```

```
employeetable = array[1..4] of record
  name : packed array[1..10] of char;
  payinfo : paytype;
end;
```

```
const
  workers = employeetable(
    ('Charlie ',(clerk,paid, 3.40)),
    ('Samuel ',(indian,paid,5.25)),
    ('Maxine ',(chief,paid, 6.85)),
    ('Edward ',(president, unpaid))
  );
```

### Reset of an undefined file, Page 61

**Reset** of a nonexistent file generates an error message. In versions of Pascal-2 previous to 2.0J, the **reset** would create an empty file.

### Emt procedure, Page 64

Predefined **emt** procedure is valid only for RSTS/E systems; the RSX manual should have contained a note to this effect.

### Installation Guide, Page 173

PROCRF: should be PROCRE:

### Installation Guide, Page 175

The Installation Control File (Appendix A) contains the line:

```
;      DEBUG.OLB
```

This line should be:

```
;      DEBUG.ODL
```

## Pascal-2 RT-11 manual

### Emt procedure, Page 60

The predefined **emt** procedure is valid only for RSTS/E systems; the RT-11 manual should have contained a note to this effect.

### Prose Guide, Page 132

The guide should note the restriction that Prose is too large to run under the XM monitor.

### Installation Guide, Page 170

The command:

```
ASSIGN SY: DK:
```

should be:

```
ASSIGN SY DK
```

RT-11 V4 users can use the COPY command in place of RPIP to copy the Pascal files, e.g.:

```
.COPY
From? MTO:*. *
To? SY:*. *
.COPY LIBFPP.OBJ PASCAL.OBJ
.COPY SJ.SAV PASCAL.SAV
```

Similarly, on V4, the DELETE command can be used to clean up the system disk after installing the Pascal system. The format is:

```
.DELETE SJ.SAV,XM.SAV
```

See the Sample Installation Commands for details.

## Pascal-2 RSTS/E manual

### System error procedure, Page 33

The section should end with this note:

The error procedure forms part of the Pascal run-time system that is shared by all RSTS/E users and is not normally replaced on a system-wide basis. Programs requiring the replacement of the error procedure must be compiled as separate run-time systems, by means of the /rts switch.

### Emt procedure, Page 57

The predefined **emt** procedure is defined in the Language Specification, but a longer section with several examples was left out of the Programmer's Guide. This section is being sent to RSTS/E users.

## All Pascal-1 manuals

### Identifiers and MACRO instructions

If a program contains embedded assembly code, no identifier in that program can have the same name as a MACRO instruction. For example, a constant named **mov** will conflict with the embedded code **mov RO, -(sp)**. In addition, a function or procedure that has the same name as a MACRO instruction and is declared as external will generate an error when assembled.

For example, this program creates a MACRO-11 error:

```
function sp:boolean;external;
```

```
begin
  if sp then writeln('It works');
end.
```

### Exiting from the Debugger

The **q** command or a Control-Z (^Z) will exit from the Debugger.

## Pascal-1 RSX manual

### I/O Control Switches, Programmer's Guide

Add this switch:

**/buff:n** (Buffersize): Pascal-2 normally allocates the minimum space required for a file buffer, which is usually 512 bytes but is dependent on device and file characteristics. More efficient I/O transfers can be performed at the cost of additional memory. The **/Buff:n** switch specifies the storage (in decimal bytes) to be allocated to a file buffer.

### Overlays, Programmer's Guide

The correct line in the ODL example should be:

```
.ROOT SINGLE,SYSIO,*MAIN-*(SUB1,SUB2,SUB3)
```



## Pascal-1 RT-11 manual

### Installation Guide

The Pascal-1 installation program for RT-11 requires a machine with at least 24KW (48KB) of user memory. This causes problems on small RT-11 SJ systems and on all TSX systems. The problem occurs while the installation program copies files from the distribution medium to the system disk. Circumvent the problem by copying the required files manually. This restriction will be fixed in the V1.2K release of Pascal-1.

## Oregon Software

The Pascal Newsletter, copyright 1981, Oregon Software, Inc.  
ALL RIGHTS RESERVED.

Printed in USA

RSTS, RSX, RT-11, AND IAS are trademarks of Digital  
Equipment Corp.

2340 SW Canyon Road, Portland, Oregon 97201

# Pascal NEWSLETTER

NUMBER 6

OREGON SOFTWARE

SPRING 1983

## ***Pascal-2 Version 2.1 to be released***

Oregon Software now has Version 2.1 of our optimizing Pascal compiler in field test for RSX, RSTS, and RT-11 systems. Version 2.1 for all systems will be released to general users June 1, beginning with RSX.

The upgrade from V2.0 represents a major revision of the Pascal-2 software, to include conformant array parameters, improve error diagnostics, enhance performance, remove size limitations on user programs, and fix all major bugs collected in the last year.

Immediately after field test, Version 2.1 will be released to all customers who have reported bugs since Version 2.0K was released last year. These shipments will probably be concluded before June 1. Then, the V2.1 update will be released to all supported customers who send us a written request.

With the energy that has gone into producing V2.1, we did not release any interim updates after V2.0K. For this reason, we will send Version 2.1A to any customer who renewed support after the release of V2.0K, even if that customer's support expires before the release of V2.1A. Updates are free on floppies and magtape; a fee is charged for other media. All customers must request V2.1 in writing.

Because many new features have been added and because the code generated by the V2.1 compiler is different from that of V2.0, users will need to modify and recompile existing Pascal-2 programs once they have installed V2.1. Users may wish to redesign existing programs to take advantage of the new features such as conformant array parameters, lazy I/O and user control of run-time error reporting.

With the addition of conformant array parameters, the Pascal-2 compiler conforms to Level 1 of the proposed ISO standard (ISO dp7185). With conformant array parameters, users can write general procedures that accept arrays with different lower and upper bounds. With V2.0, users have to write a new procedure for each array that varies in size or bounds.

We have substantially improved the error diagnostics by including a procedure walkback generated as a result of a run-time error. The walkback displays the error message, the point of error in Pascal source terms, and a traceback of procedure calls leading to the error. Version V2.0 has no equivalent run-time diagnostics.

In addition, a new I/O error trapping capability allows users to recover from I/O errors with their own code, to change the wording of run-time error messages and to print additional information besides that printed in the walkback.

Other substantive changes include: the addition of "lazy I/O," a scheme by which data is not read or written until actually used in the program; elimination of certain size restrictions, allowing larger programs to be compiled; capability to include more procedures and type definitions; new procedures to do common and useful I/O procedures, including delete and rename; and features to help users of small systems.

Changes and additions to V2.1 are documented in a 70-page supplement to the *Pascal-2 User Manual*. The supplement will be supplied with all updates. The changes will be included in a new edition of the manual to be printed in the late spring.

This newsletter carries a summary of the major changes, plus articles on the use of various new features. Major bug fixes are described in the Bug Log.

---

### **In this issue...**

|                                                    |         |
|----------------------------------------------------|---------|
| Pascal-2 Version 2.1 to be released . . . . .      | Page 1  |
| UNIX version joins Pascal-2 family . . . . .       | Page 2  |
| SourceTools available for RSX . . . . .            | Page 2  |
| Summary of changes/new features for V2.1 . . . . . | Page 3  |
| Terminal I/O . . . . .                             | Page 4  |
| Single-character I/O . . . . .                     | Page 4  |
| FORTRAN carriage control . . . . .                 | Page 5  |
| I/O error trapping . . . . .                       | Page 6  |
| Random access to text files . . . . .              | Page 8  |
| Lazy I/O . . . . .                                 | Page 10 |
| Converting unsigned integers . . . . .             | Page 11 |
| The Log: Pascal-1 and Pascal-2 . . . . .           | Page 12 |
| Information exchange . . . . .                     | Page 13 |
| OPUS Communiqué . . . . .                          | Page 14 |
| Marketing group evolves . . . . .                  | Page 15 |
| User manuals for V2.1/1.3 . . . . .                | Page 16 |



## ***UNIX version joins Pascal-2 family***

Oregon Software has released the Pascal-2 compiler on the UNIX operating system for the PDP-11. The new product includes an enhanced version of our interactive source-level debugger.

UNIX, originally developed by Bell Laboratories, is sold through license agreements with Western Electric. Popular in university environments originally, UNIX is becoming a major force in software development applications and has spawned many UNIX "look-alike" operating systems.

A key feature of UNIX is the collection of supporting "tools" that provide a software developer with standard, easy-to-use programs to help with coding and development. Oregon Software's Pascal compiler system contains similar kinds of tools — debugger, execution profiler, and development utilities.

The debugger allows the programmer to work interactively at the source level to solve logic errors in applications, thus

simplifying and speeding development. The debugger runs as a separate process from the application code that is being debugged, keeps track of multiple compilation units, and performs breakpoint debugging without slowing down the application code.

The profiler aids the applications programmer by identifying execution bottlenecks. With data supplied by the profiler, the developer can reorganize portions of the code to substantially improve overall execution speed.

The Oregon Software Pascal-2 UNIX compiler supports all capabilities of standard Pascal and conforms to the draft proposed Pascal standard (International Standards Organization dp7185.1, level 0). As a result, users are assured that code developed under Pascal-2 can be moved to other systems with standard Pascal compilers.

Pascal-2 for UNIX is available now directly from Oregon Software or from authorized distributors.

## ***SourceTools available for RSX***

Oregon Software is now releasing SourceTools for RSX, including VMS in compatibility mode, to general end users. The seven-program package helps programmers develop and coordinate large software projects. SourceTools has been in field test for the last three months and no major problems have been reported.

The source-control system, SourceCon, is the foundation of the package. SourceCon monitors files placed under its control, to prevent programmers from making simultaneous and conflicting changes and to keep a development history of the source files. In a sense, SourceCon provides a "library service" for programmers and writers, recording changes to text files and the reasons for them, and ensuring that earlier versions of a module are always available.

Another program in the SourceTools package simplifies and automates the task of rebuilding software from component parts. The SourceTools MAKE program automatically determines which files are out of date, then executes only the commands needed to rebuild those out of date files.

In a more conventional environment, the programmer must remember how all the components of a large program fit together whenever some components are modified, and must recall exactly how to re-create each component so that it fits with others in the module. This is the old "recompile everything in sight" problem. Or, the programmer has to keep a set of command files around for each of the various parts and for combining the parts together.

Two additional utilities alleviate the tedium of maintaining parallel or alternate versions of files. A text comparison program, TEXTCOM, isolates differences between two text files. A stream editor, SEDIT, interprets a text-editor script and applies the editor-commands to its input text file. By a switch in the text comparison program, TEXTCOM can output an editor script compatible with SEDIT. This combination can be a powerful tool for maintaining parallel files on remote systems and for cross-development work.

These time-saving software tools are used extensively by our own programmers.

# Summary of changes and new features for Pascal-2 Version 2.1

New features provide the following benefits for users:

- Conformant array parameters allow you to write general procedures that accept array parameters of different sizes and with different lower and upper bounds.
- Non-decimal constants allow you to use constants in number bases ranging from base 2 (binary) to base 16 (hexadecimal).
- Run-time error walkback aids in diagnosis of run-time errors.
- User-processing of run-time errors gives you control of run-time error reporting. You can change the wording of run-time error messages and print additional information besides the walkback. The walkback may be disabled with the **nowalkback** switch.
- I/O error trapping capability allows you to recover from I/O errors with your own code. Article on page 6.
- A new procedure provides an explanation of I/O errors (RSTS and RSX only). Article on page 6.
- Lazy I/O. A new I/O scheme in which data is not read or written until actually used in the program. Article on page 10.
- New I/O control switches extend the capabilities of the file system. See related article on page 10.
- New procedures **getpos** and **setpos** simulate random access to files of type **text**. Article on page 8.
- New syntax of **%include** directive allows you to specify the disk volume number, UIC and version number of the included file.
- New built-in procedures **rename** and **delete** allow you to rename or delete files from within the program.
- New function **space** determines the amount of stack and heap available to an executing program.
- Eight-bit characters allow you to use extended character sets.
- **WK**: specification allows you to specify the device to which the compiler directs its working data files and helps small-systems users reduce load on the default disk.

The following changes have been made:

- Certain size restrictions have been eliminated to allow larger programs to be compiled. Also, V2.1 programs can have more procedures and more type definitions than V2.0 programs.
- New run-time error numbers and messages have been added.
- Single-character I/O has been added to RSX. Article on page 4.
- New method of reading MCR command lines has been added to RSX.
- New ways to overlay Pascal-2 programs; new way to overlay the Pascal-2 support library.
- New support library interface. Library entry points have been changed to the form **P\$nnn**, where **nnn** is a number in the range 0..135.
- Procedure **timestamp** added to RSX. Returns the date and time.

## Pascal-1 V1.3 in field test

Version 1.3 of Pascal-1 is also in field-test for RSX, RSTS, and RT-11 systems. V1.3 also will be made generally available to end users by June, along with Pascal-2 V2.1.

V1.3 will include lazy I/O, a new library interface, and a new initialization interface. Pascal-1 users will be able to use the new I/O error diagnostics, but they must change their source programs.

A number of bugs also have been fixed.

## Terminal I/O for RSX

For each operating system, the terminal driver acts as an intermediary between a Pascal program and the terminal that is running that program. The terminal driver is record-oriented, which means it sends a full line of text to the screen or to the program instead of sending one character at a time.

For example, when a `read` statement reads input from a terminal, the terminal driver reads each character and places it in an internal buffer until the terminal driver encounters an end of line (a carriage return). At this point, the terminal driver sends the entire line to the program and lets the program process the line one character at a time.

The same is true for `write` statements, except that the terminal driver buffers the characters being sent to the terminal until a `writeln` clears the buffer or until the buffer size is reached. Then the record is displayed on the screen. The size of the buffer can be controlled with the `buffersize:n` file control switch. See "Single-Character I/O for RSX."

As mentioned above, the `writeln` statement instructs the terminal driver to send the output to the screen. After the terminal driver prints a line, it positions the cursor over the first character of the line it just printed. The usual output sequence sent by the operating system's terminal driver is: line feed, data, carriage return. This sequence moves the cursor to the next line, prints the data and returns the cursor to the first position of the newly printed line. This method of writing lines to a terminal is different from other operating systems where the normal output sequence is data, carriage return, and line feed (print the record, move to the first character of the line, then move to the next line).

The normal sequence of commands issued by the terminal driver may not particularly suit special I/O applications such as direct cursor addressing. On RSX, for example, the line-feed and carriage-return characters are often not needed or are sometimes unwanted. To gain control of the terminal-I/O command sequence, use the `ftn` file control switch, a feature which allows you to override the effects of the terminal driver (see "Using FORTRAN Carriage Control under Version 2.1 for RSX").

## Single-character I/O added for RSX

The `buffersize:n` I/O control switch sets the line size of a text file to `n` bytes. By setting your terminal's internal line

size to 1 with `buffersize:1`, you can write programs that perform single-character input and output. This feature is useful when you need to do special output formatting to a video terminal, such as direct cursor addressing of output on the screen.

### CAUTION

Single-character input/output increases system overhead and should be used with care. Overhead is increased because the monitor must be called for each character read from and written to the terminal. Several users simultaneously outputting in single-character mode can significantly reduce the system response time.

To enter single-character mode, specify the `buffersize:1` switch on the `reset` statement for single-character input and on the `rewrite` statement for single-character output. In single-character mode, the terminal driver reads and writes the characters as usual. But since the internal buffer size is one byte, each new character fills the buffer, causing the buffer to be emptied. In other words, each character written via `write` is immediately printed on the screen; each character that you enter is immediately sent to the program.

Continued on Page 8

```

program Single;
var
  Ch: char;

begin
  reset(input, 'TI:/buff:1');
  rewrite(output, 'TI:/buff:1');
  write('Type a message: ');
  while not eoln do
    begin
      read(Ch);
      write(Ch);
    end;
  end.

>RUN SINGLE
Type a message: ddoouubbllee_vviissioonn
>

```

Program Single uses a combination of single-character input and single-character output procedures under the RSX operating system. When compiled and linked and run, the program simply echoes each entered character as shown.



## Using FORTRAN carriage control under Version 2.1 for RSX

Pascal-2 allows your Pascal programs to write **text** files that follow the FORTRAN standard output conventions. To do this, specify the **ftn** file control switch on the **reset** or **rewrite** statement that opens the file. For interactive I/O, the **ftn** switch is used with the standard file **output**.

FORTRAN conventions state that the first character of each line of an ASCII file is nonprintable and is used to control the vertical formatting of an output file or terminal screen (see table of commonly used vertical formatting characters). Most characters have no useful meaning in the first position; however, some characters significantly affect the format of the output. The *RSX-11M/M-Plus I/O Drivers Reference Manual* contains a complete list of these vertical formatting characters.

The null character (**chr(0)**) can be used to prevent the terminal driver from adding any special characters to the data you write to the terminal. You will have to insert the carriage-control characters yourself, as shown in the last line of sample program **Ftn**. In the output shown for program **Ftn**, the "overprint" **writeln** replaces the characters "Normal output" from the previous **writeln** with "Overprint\*\*\*\*," resulting in a different line of text. If you set your terminal to a slow baud rate, you would actually see the overprinting occur. If you direct the file to a printer, the overprinted line will contain overstrikes.

The **ftn** switch does not solve all terminal I/O problems. The data is not written to the terminal immediately; the characters are still buffered until a **writeln** is executed. You can overcome this problem by using the feature that allows single-character I/O.

```
program Ftn;

begin
  rewrite(output, 'TI:/ftn');
  writeln(' Normal output');
  writeln(' More normal output');
  writeln('ODouble space');
  writeln('1Page eject');
  writeln(' Normal output of a long line');
  writeln('+Overprint****');
  writeln('$Prompt output: ');
  writeln(chr(0), 'Internal format',
          chr(13), chr(10));
end.
```

>RUN FTN

Normal output

More normal output

Double space

<ff> ————— skips to top of next page

Page eject

Overprint\*\*\*\* of a long line

Prompt output: Internal format

>

**Program Ftn** shows the use of FORTRAN standard output conventions in a Pascal program under RSX. The first character written on each line is interpreted as the vertical-format control character. **Chr(13)** is the carriage-return character, and **chr(10)** is the line-feed character. The output shown appears on your terminal when the program is run.

### Commonly Used Vertical Formatting Characters

| Character | Meaning               | Output Sequence                       |
|-----------|-----------------------|---------------------------------------|
| <space>   | Normal output         | Line feed, data, carriage return      |
| 0         | Double-space          | Two line feeds, data, carriage return |
| 1         | Page eject            | Form feed, data, carriage return      |
| +         | Overprint             | Data, carriage return                 |
| \$        | Prompt                | Line feed, data, remain on same line  |
| <null>    | No special formatting | Data only                             |

## New routines trap I/O errors

Pascal-2 V2.1 permits you to write programs that trap and detect many kinds of I/O-related (normally fatal) errors. Using three predefined routines to process I/O errors with your own code, you can terminate the program at the occurrence of an I/O error or continue execution in spite of the error. You can print your own diagnostics with a fourth procedure.

The three error-trapping routines — procedure **noioerror** and functions **ioerror** and **iostatus** — are predefined and do not need to be declared in your program. They accept a file variable as their only parameter. The **sayerr** procedure, which prints the text of a given error message, is not predeclared, so you must declare it when used. Although details vary slightly according to the operating system, these routines work as follows.

Procedure **noioerror** specifies that the calling program will handle any I/O errors that result from reading or

writing to the specified file. The file must be open before **noioerror** is called.

Function **ioerror** determines the status of the last I/O operation that the program performed on the specified file. This **boolean** function returns a **true** value if an I/O error has occurred or a **false** value if the operation was successful.

Function **iostatus** helps your program determine the cause of the error by returning an integer error code describing the last attempt to access a file. Your program can either bypass the problem and continue processing, or terminate so you can correct the problem.

You can pass **iostatus** as a parameter to **sayerr**, an **external** procedure which prints the text of the error message corresponding to the value returned by **iostatus**.

Continued on Page 7

## Printing I/O errors under RSX

Procedure **sayerr** prints the text of a given error message. A negative error code indicates that the error is specific to the RSX operating system. A positive error code indicates that the error is detected by the Pascal-2 support library. Pascal-2 error codes, along with the text of the error message and a brief explanation of the cause, are listed in the V2.1 supplement to the *Pascal-2 User Manual*. RSX I/O error codes are listed in the *RSX I/O Operations Reference Manual*.

As an external procedure supplied in the Pascal support library, you must declare the **sayerr** procedure in your program as shown:

```
procedure Sayerr(Status: integer);
external;
```

where **Status** is the error code of the error.

Procedure **sayerr** takes an RSX I/O error code (a negative number) and looks in the file **LB:[1,2]QIOSYM.MSG** to print the text of the error message. **Sayerr** only prints messages for negative I/O error codes in the range -255..-1; **sayerr** ignores error codes that lie outside this range, printing nothing.

```
program Opnerr;
var
  F: text;
  Status: integer;

procedure SayErr(Code: integer);
external;
begin
  reset(F, 'XXXX:', 'test.dat', Status);
  if Ioerror(F) then
    begin
      writeln('I/O status=', Iostatus(F));
      SayErr(Iostatus(F));
    end;
end.

>RUN OPNERR
I/O status=      -55
Bad device name  _____ message sayerr prints
>
```

Program **Opnerr** shows the use of procedure **sayerr** on RSX. The program attempts to open a file called **TEST.DAT** on a fictitious device with the name **XXXX:**. Normally, this program would fail with no specific indication of what caused the **reset** failure. The error is detected by **Ioerror**. Function **iostatus** returns the value -55 and passes the parameter to **sayerr**. The call to **sayerr** prints the text of the message for RSX I/O error code -55: "bad device name." When this program is compiled and executed on RSX, it yields output similar to that shown above. A RSTS example produces similar results.

```

program Iotest;

var
  I, Times: integer;

begin
  Noioerror(input);
  for Times := 1 to 5 do
    begin
      write('Type an integer: ');
      read(I);
      if Ioerror(input)
        then writeln('Error detected. Status=', Iostatus(input))
        else writeln('The integer was: ', I: 1);
      readln;
      writeln;
    end;
  end.

```

**>RUN IOTEST**

Type an integer: 1234

The integer was: 1234

Type an integer: 123456789 \_\_\_\_\_ integer too large  
 Error detected. Status= 19 \_\_\_\_\_ Pascal-2 error code for invalid integer

Type an integer: ffg \_\_\_\_\_ non-integer characters  
 Error detected. Status= 19 \_\_\_\_\_ Pascal-2 error code for invalid integer

Type an integer: 77  
 The integer was: 77

&gt;

**Program Iotest** illustrates the use of the use of pre-defined error-trapping routines for RSX. This program continues to execute after an I/O error is found; without these routines, the first error would cause the program to abort. The call to **noioerror** notifies the run-time system that the program will handle errors detected on the standard file **input**. When compiled and run, the results of such a program are similar to the above. The first entry results in a successful read of the integer **I**. The second and third entries result in a Pascal-2 run-time error. The final entry is successfully read, and the program ends. RSTS and RT examples produce similar results.

When you call these routines, you are responsible for checking the status of each I/O operation, to ensure that it was successful. If you fail to check the status afterwards, the results will be unpredictable.

The I/O error-trapping procedures can be used to determine the reason that a file could not be opened. To use this feature, specify the fourth parameter on calls to **reset**

and **rewrite**. The use of the fourth parameter keeps the **reset** or **rewrite** from trapping a normally fatal "open" error. This allows your program to recover and continue, or terminate under your control. (Sample program **OPNERR** demonstrates use of this feature under RSX.)



## Random access to text files simulated

Because lines of text vary in length, the **seek** procedure cannot predict the location of a line within a text file. The Pascal-2 support library for V2.1 supplies two external procedures, **getpos** and **setpos**, that simulate random access to **text** files. Working together, these procedures use a set of values to locate the next line of text. **Getpos** determines the starting location of the line and **setpos** returns the file pointer to the specified starting location of a line within the file.

For text files, the beginning of each line is denoted by a block number and a byte offset into that block. Each block contains 512 bytes. The first line of a file starts with block 1, offset 0. If you try to access a nonexistent position or a position in the middle of a line, an I/O error will result. When the file is read, use **getpos** to determine the starting position of the next line, and save that block and offset combination for later use by **setpos**. The block number and byte offset must be values returned by **getpos**. You cannot compute the values yourself.

These two procedures are not predefined and must be declared in your program as **external**. These procedures do not provide "true" random access; you cannot use them to access a file you have not previously read.

Sample program REVERSE shows the use of **getpos** and **setpos**.

### 'Getpos' procedure

Procedure **getpos** determines the starting position of the next line to be read from or written to a text file. **Getpos** requires three parameters, which are passed by reference, as shown below. Together, the parameters **Block** and **Offset** point to the next line to be processed.

```
procedure GetPos(var F: text;
  var Block, Offset: integer);
  external;
```

where

- F** is the file variable of type **text**.
- Block** is the returned disk block number of the next line in file **F** to be read or written.
- Offset** is the returned byte offset into **Block**.

You should always call **getpos** to obtain the location in the file before you call **setpos**, so the block and offset values being passed to **setpos** are valid.

### 'Setpos' procedure

Procedure **setpos** positions the file pointer to a specified block number and byte offset into that block. **Setpos** accepts the same three parameters as **getpos**, except **Block** and **Offset** are passed by value. The **setpos** declaration is:

```
procedure SetPos(var F: text;
  Block, Offset: integer);
  external;
```

where

- F** is the file variable of type **text**.
- Block** is the block number to which the file pointer is set.
- Offset** is the byte offset into **Block**.

Together, **Block** and **Offset** point to the new position. To stress an earlier point, the block number and byte offset must be values returned by **getpos**. Do not attempt to compute the values yourself. Save the returned values for later use.

### Errors

If an error is detected while **setpos** tries to position the file, the end-of-file flag **eof** is set to true. The **ioerror** and **iostatus** support library procedures may help you to determine the reason that the line could not be accessed. (For details on **ioerror** and **iostatus** see article on page \_\_.) If a file is positioned to a block and offset that does not correspond to the first character of a line, the results will be unpredictable.

### Using FORTRAN... Continued from Page 4

No special formatting characters are inserted when you use the **write** statement (see "Using FORTRAN Carriage Control under Version 2.1 for RSX"). A **writeln** statement will print a carriage return followed by a line feed, as if the buffer were empty. You can supply formatting characters by using the **chr** function to generate the appropriate characters.

For single-character input, you do not need to type a carriage return after each character to signify the end of the line.

```

program Reverse;

{ Print the contents of a file in reverse line order }

type
  Pointer = ^position;
  position =
    record
      Next: pointer;
      Block: integer;
      Offset: integer;
    end;

var
  F: text;
  Filename: packed array [1..80] of char;
  P, X: pointer;

procedure GetPos(var F: text; var Block, Offset: integer);
  external;

procedure SetPos(var F: text; Block, Offset: integer);
  external;

begin
  write('File name? ');
  readln(Filename);
  reset(F, Filename);
  P := nil;
  repeat
    { read the file }
    new(X);
    with X^ do GetPos(F, Block, Offset);
    X^.Next := P;
    P := X;
    readln(F);
  until eof(F);

  while P <> nil do
    { write the file }
    with P^ do
      begin
        SetPos(F, Block, Offset);
        if not eof(F) then
          begin
            while not eoln(F) do
              begin
                write(F^);
                get(F);
              end;
            writeln;
          end;
        P := P^.Next;
      end;
  end.

```

**Program Reverse** demonstrates the use of the **getpos** and **setpos** procedures to simulate random access of text files on RSX, RSTS, and RT. The program reads a text file and saves the position of each line in a linked list. It then prints the file in reverse line order so that the last line of the file is printed first, the next-to-last line is printed second, and so on, with the first line printed last.

## Lazy I/O interface for all systems

For standard Pascal, an interactive input file, such as a terminal, poses a problem. A program must always be able to determine the current status of an open file, i.e., it must be able to retrieve the current record from the buffer variable (F\*) and to check the current values of `eoln` and `eof`. Since an interactive file is being created as it is being read, the program must periodically wait for additional input to determine the file's current status. With the Pascal support library's new approach to interactive I/O, the program should not wait for input at inconvenient times.

Pascal-2 Version 2.1 uses an input interface known as "lazy I/O" to handle input from `text` files. Since a file's status needs to be defined only when the program actually refers to it, lazy I/O safely delays any input operation until the program uses its results. When a Pascal program requests an input operation on a text file, the operation is recorded for later use. The delayed operation is triggered by any subsequent reference to the file's buffer variable, `eof` value, or `eoln` value. The delay is invisible to the program but is visible to the user from the way the program is synchronized with interactive input.

To use lazy I/O, you need to be aware of its effect on synchronization of input and output operations. As an example, consider a simple program that reads its standard file `input`, which is connected to a terminal. The program prompts for each line and stops at the end of the file. The design of the program is dictated by two requirements:

- For the prompt to be effective, it must appear before the user is required to type the line.
- To detect the end of the file correctly, the program must check for it before reading each line.

To meet both of these requirements, the program must print the prompt before performing any operation that requires the next line of the file to be known: e.g., checking for "end of file" or reading the line. The sample programs in the "Conversion Note" show the differences in design between Version 2.0 and 2.1.

### CONVERSION NOTE

Lazy I/O changes the compiler's implementation of interactive I/O. Programs compiled with Pascal-2 Version 2.0 may have to be revised to conform with the new schema. For example, under V2.0 program `INTERACTIVE` is coded as follows.

```
program Interactive(input, output);
{sample for version 2.0}

begin
  while not eof do
  begin
    write('prompt:');
    readln;
  end
end.
```

The program above compiles and executes under Pascal-2 V2.1, but the program is not synchronized with input from the terminal. As a result, you do not see the prompt until you type in a line. In other words, you are prompted for the line you have just entered. For the prompt to be effective, program `INTERACTIVE` should be coded as follows.

```
program Interactive(input, output);
{sample for version 2.1}

begin
  write('prompt:');
  while not eof do
  begin
    readln;
    write('prompt:');
  end
end.
```

**NOTE:** Under the Digital operating systems, typing Control-Z (~Z) in response to the prompt signals an "end of file" on the interactive input file, halting the program.



## Converting unsigned integers on the PDP-11

PDP-11 computers store integer variables in 16-bit words. These 16-bit words may be interpreted as signed or unsigned integers. As a signed number, in two's complement notation, 16 bits can represent numbers in the range of -32767..32767. When considered as unsigned numbers, a word can represent an integer in the range of 0..65535. For both signed and unsigned integers, the bit patterns in the word are the same; only the interpretation of the bit patterns differs.

When their values are compared or used in mathematical expressions, unsigned integers differ greatly from signed integers. As an example, consider a word in which all 16 bits are set to one. This word has a value of -1 when interpreted as a signed integer, or a value of 65535 interpreted as an unsigned integer. When this word is compared with some other value, the PDP-11 uses different combinations of instructions for signed and unsigned comparisons. If this number is multiplied by two, the result is a value of -2 for signed or 131070 for unsigned. The latter is an overflow condition because the result will not fit within 16 bits.

The Pascal-2 compiler and support library also differ in their treatment of signed and unsigned integers. The compiler can deal with unsigned integers, but the library contains a single routine that interprets all integers as signed values. For example, if you define a variable to be of type **integer** in your Pascal program, the compiler treats that value as a signed integer, unless you specify an unsigned integer using a subrange notation such as:

```
type
  unsigned = 0..65535;
```

```
var
  X: unsigned;
```

According to your data declarations, the compiler generates the correct code to compare, multiply, or divide unsigned numbers. However, the support library only prints out a signed integer unless you use the following procedure in your program instead of the **write** statement:

```
procedure Uwrite(X: unsigned;
                 Width: integer);

var
  k: integer;

begin
  if (X > 32767) and (Width >= 0) then
    begin
      if Width > 0 then
        Width := Width - 1;
        write(X div 10: Width);
        X := X mod 10;
        Width := 1;
      end;
      write(X: width);
    end;
end;
```

This procedure takes an unsigned integer and a field width as its parameters. The number is printed as a value in the range of 0..65535, right justified in the specified field.

The PDP-11 floating-point hardware uses a signed conversion when it converts an integer value to a real value. If you wish to convert an unsigned integer to a real number, you should use the following function.

```
function Ufloat(X: unsigned): real;

var
  R: real;

begin
  R := X;
  if R < 0.0 then
    R := R + 65536.0;
  Ufloat := R;
end;
```

This function takes an unsigned integer as its parameter and returns a real value in the range of 0.0 to 65535.0. The **trunc** and **round** procedures convert real numbers to integers. The floating-point hardware assumes a signed conversion, so the following function should be used when an unsigned integer result is desired:

```
function Utrunc(R: real): unsigned;

begin
  if (R > 65535.0) or (R < 0.0) then
    writeln('Unsigned number out of range');
  if R > 32767.0 then
    R := R - 65536.0;
  Utrunc := trunc(R);
end;
```

This function takes a real number in the range 0.0 to 65535.0 and converts it to an unsigned integer. The unsigned **round** function is very similar to the above unsigned **trunc** function.

# ***The Log: Pascal-1 and Pascal-2***

Pascal-2 V2.1 fixes a number of bugs, including many that generated obscure or random behaviors. Several fixes listed for V2.1 involve the support library; such fixes also apply to V1.3.

## **Pascal-2**

V2.1 fixed the bugs consistently generating these symptoms:

### **/EIS, /FIS Arithmetic**

Real operations under `/eis` and `/fis` generated incorrect results at times. Problem occurred in the handling of intermediate, stacked results of the simulated real arithmetic or (`fis` instructions) operations.

### **Packed Records**

Packed records would generate incorrect results if the record was moved to the odd byte of the word.

### **Odd Address Traps**

Several problems, resulting in odd-address traps at run time, were fixed.

### **Odd Address Traps During Set Operations**

Compiler assumed that sets were always word-aligned (true unless the set is only one byte long). And it didn't ensure that the two operands were unpacked.

### **Illegal use of MOD**

Illegal use of `mod` caused a compiler trap. An error message is now generated.

### **'%Include' Syntax Checking**

Previously, the compiler didn't catch errors in syntax. It now generates the proper error message.

### **'Repeat .. Until' Errors**

Errors were occasionally generated in `repeat .. until` operations.

### **Expression Evaluation Errors**

During optimization, the compiler would sometimes evaluate

an expression when it was not safe to do so.

### **Boolean Assignments in Debugger**

If the first Debugger command in a program being debugged involved assigning a value to a boolean variable, the results would be "out of range."

### **Debugger 'L' Command**

The Debugger L command was not ignoring enough lines at the top of each listing page; part of the page header was being included in the listing of the code.

### **Debugger, File Pointers**

The Debugger had problems with pointers to files, either printing the wrong value or trapping.

### **Debugger, Packed Records**

The Debugger was not making the transition between unpacked and packed parts of a record properly. Incorrect values were printed when individual fields were accessed.

The Debugger treated a packed array of unsigned bytes as if the array elements were signed, causing elements larger than 127 to print as negative.

### **Debugger, Real Constants**

The Debugger gave incorrect results in writing the value of real constants.

### **/FTN Switch**

When files were opened with the `/ftn` switch, the second character on the line was being used to position the cursor on CRTs. The fix moves the carriage-control character to a register to be sure that the upper byte is zero. A solution for users with previous versions is to write a null character (`chr(0)`) as the second character on the output line, immediately following the FORTRAN control character.

### **Text Files and 'Put'**

Creation of a text file with `put` operations, rather than `write` or `writeln`, altered one extra byte past the end of the allocated buffer. This would zap whatever data happened to be on the heap after the file buffer. The overflow was detected, but the heap was damaged.

## Pascal-1

V1.3 bug fixes are currently in progress. A summary of the fixes will be included with the update release. Two fixed thus far are:

### Abs() problem

The **abs()** function gave incorrect results for **real** operations. It now gives correct results for both reals and integers.

### Trapping on underbars

A program containing underbars in identifier names caused the compiler to trap. It now gives an error message.

## Pascal NEWSLETTER

OREGON SOFTWARE

2340 SW Canyon Road  
Portland, Oregon 97201

Collins Hemingway, editor

David Spencer and Michael Kuhn, staff writers

The Pascal Newsletter is published quarterly by Oregon Software, Inc., 2340 SW Canyon Rd., Portland, OR 97201; (503) 226-7760. Each customer of Oregon Software receives one free subscription per site. Additional subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor. We pay \$100 per newsletter page for any article we print.

Copyright © 1983 by Oregon Software, Inc.  
ALL RIGHTS RESERVED.

RSTS, RSX, RT-11, PDP-11, VAX/VMS, and IAS are trademarks of Digital Equipment Corp. UNIX is a trademark of Bell Laboratories. MC68000 is a trademark of Motorola, Inc. Pascal-1, Pascal-2, SourceTools, and Pascal Newsletter are trademarks of Oregon Software, Inc.

Printed in USA

## Information exchange

If you need information on technical applications involving Pascal, or if you have an application that might interest other users, send us a brief description for inclusion in the Information Exchange. Your description should follow the format of the items below. Interested parties can contact one another directly.

**Data Base** written in standard Pascal. Intended as an educational tool and currently used by computer science classes, the working system consists of four interactive programs, with a screen updating program and a calculation routine scheduled for inclusion this summer. For information, contact: Leon Schilmoeller, Computer Science Dept, Augustana College, Sioux Falls, SD 57102, (605) 336-5495.

**Pascal Users' Group.** The new contact person for the PUG is: Charles Gaffney, 2903 Huntington Rd., Cleveland, Ohio 44120.

**Pascal Programmer** sought by dynamic, research-oriented company. Ideal candidate should have a thorough knowledge of Pascal programming and be familiar with PDP-11 minicomputers and/or the Motorola MC68000 microprocessor. Send letter of inquiry and resumé to: Personnel Director, P.O. Box 1201, Portland OR 97207.

**Compiler Writer** wanted. Dynamic, research-oriented company seeks senior software engineer with a thorough knowledge of Pascal programming and an extensive background in writing code generators. Send letter of inquiry and resumé to: Personnel Director, P.O. Box 1201, Portland OR 97207.

---

# OPUS Communiqué

*Oregon Pascal Users Society*

---

TUSTIN, California — As of February, Oregon Pascal User's Society (OPUS) has more than 100 members. At the first two OPUS meetings held in Irvine, California, the 10 attending members of the group decided that the organization should be structured as two major subdivisions: United States and International.

Within the United States, OPUS will be divided into subgroups by time zone, e.g., West Coast, Mountain, Central, and East Coast. The International OPUS will be subdivided by geographical proximity. The international group is not yet subdivided because of the small number of international members (so far) and their extreme geographic diversity. Canada, with 10 to 15 members, is an exception.

Many local OPUS (LOPUS) chapters can be formed within each time zone. Now, we need people in the local areas who will accept the responsibility of organizing a local OPUS. The Southern California OPUS (SCOPUS) is an example of the local OPUS structure that most members felt would be useful in attacking the problems of certain tasks they have encountered. The Southern California chapter held its first two meetings in January and February 1983.

Anyone interested in starting a LOPUS should contact Bruce Williams, the OPUS coordinator, at the address given below. He will help you get started.

## OPUS Projects

OPUS is conducting a survey to determine the machines and compilers currently in use by its members and the tasks (business applications, systems, etc.) for which those systems are used. The questionnaire is being mailed to each OPUS member. Responses will be collected in March and April. After compiling the results, OPUS will send to each member a membership list and a copy of the results. With the results of the survey, an OPUS member can find out who in their area is using Oregon Software's Pascal and in what applications. If you aren't a member or if you have not received a member survey, please write to Bruce Williams at OPUS.

We're also putting together a library of useful routines submitted by OPUS members.

## Notes from OPUS Members

From EOCOM in Southern California — Recently, Lyle Norton discovered that the `time` function for RT-11 takes care of the 50/60 Hz conversion before it returns the time to the calling routine. You don't have to add your own conversion code. Isn't it nice to have things done right before you start the coding? You won't, however, find this information in the manual.

From Allergan in Southern California — Jerry Shaver has done some nice routines using character I/O on RSX-11M. Jerry said he would prepare them for the OPUS library being developed. If you're curious about these I/O routines, contact him through OPUS.

## Next Communiqué

We'll say more about the OPUS library of routines and give you some guidelines for writing code that you want to submit to the library. If you want to appeal for a solution to some nagging problem or the ways others may have attempted to do something, please write to OPUS at the address below. We'll get your appeal into the Communiqué.

Bruce Williams,  
OPUS Coordinator  
c/o EOCOM  
15771 Red Hill Ave.  
Tustin, CA 92680  
(714) 730-5051, ext. 302

---

## Editor's Note:

The User's Manual for RT-11 says that the built-in function `Time` "returns a real value corresponding to the current time of day" on page 57. We saw no need to explain in detail how the time was produced.

---



## Anne Smith retires; Pat Rau named manager

Anne Smith, Manager for General Distributors, retired on March 31. As a founder of Oregon Software, she has spent the last seven years helping the company grow and succeed, building up our distributor network and creating the sales group.

Retirement is probably the wrong word for Anne's future; redirection of her energy would be better. As Anne says, "There is so much to do that I always feel I need to rush out and do it all TODAY! That's one of the reasons for early retirement — to have the time to do it."

Travel is first on her new priority list. Anne plans a two-week cruise through the Caribbean islands, including two stops in South America. "I'm really going to tourist it," she says. Of course, she'll be doing a lot of fly-fishing on our beautiful western trout streams, and she'll be cross-country skiing in the winter, mid-week when the trails are not so crowded. She also wants to take a two-week raft trip on the Colorado River, through the Grand Canyon this year or next.

Anne wants to resume volunteer work for environmental groups such as the Nature Conservancy and Audubon Society. She used to be active in these groups, but hasn't had time during her years with Oregon Software. Her close friend Pete Pedersen, a lawyer, works as a volunteer for the ACLU and she may find something there, too.

From time to time, Anne will serve as a consultant to Oregon Software and she may fill in for vacationing sales personnel occasionally.

Pat Rau succeeds Anne as Manager for General Distributors. Pat has been with Oregon Software for two and a half years, as a general salesperson and specialist in licensing agreements. Before joining Oregon Software, Pat worked for Honeywell's Portland division.

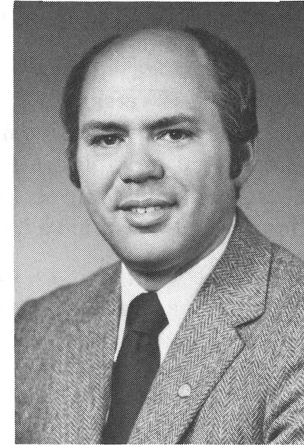


Anne Smith



Pat Rau

## David Cloutier heads marketing



David Cloutier

Oregon Software has appointed David Cloutier, formerly a software marketing manager at Intel, as vice president of marketing and sales.

David was the software marketing manager of OEM micro-computer systems at Intel's Hillsboro, Ore., division, and was product manager for the introduction of Intel's iRMX 386 operating system. Additionally, he served as chairman of the OEM division's software business planning committee. All together, David has 10 years of software marketing experience, having also worked at Zentec and Perkin-Elmer.

David expects Oregon Software to become a major software house for the MC68000, based on recent releases of a family of software products for developing high-performance applications for Motorola's MC68000. David also expects the company to continue its enhancement of products serving the DEC market, both through major upgrading of compiler products, such as the 2.1 release, and through new products such as SourceTools.

"The professional programmer is the primary focus of our work," David said. "That's because we are a company of professional programmers. Every tool we sell is a tool we first develop for our own use, and every tool we sell has to first meet our own high expectations. We must bring that quality advantage to bear in the market to give our customers the same high performance in their applications."

## User manuals revised for V2.1/1.3

New editions of all PDP-11 manuals for Pascal-2 are expected to be published in late spring. New editions of Pascal-1 manuals will be printed in the fall.

The new editions will document all new software features included with the releases of Versions 2.1 and 1.3, respectively. The Pascal-2 manuals will include new sections, expanded sections, additional examples in many areas, corrections of errors and omissions, and inclusion of all current errata. The manuals also will have indexes.

Pascal-1 manuals will be upgraded to include many of the features now found in the Pascal-2 manuals, including tutorial sections and index.

Pending the printing of the new editions, the new 2.1/1.3 features will be documented in a supplement to the current manuals. The supplement will be shipped with all updates.

User manuals for all Oregon Software products will take a new form with all new printings: looseleaf with windowed covers and binders. (You remember the binders, don't you?)

This format will allow us to send future errata as inserts, in the form of "change pages." Users can "mix and match" their Oregon Software library according to their choice of compiler and options. The manuals will now lay flat during use. The new cardboard cover allows users who preferred the bound editions to keep a semblance of that feel.

### Book Prices

As of April 1, 1983, new prices will be in effect for all books and manuals available from Oregon Software. The new price list follows:

| User Manual                                   | Type of Sale                   | Price   |
|-----------------------------------------------|--------------------------------|---------|
| Pascal-1 and Pascal-2                         | End User                       | \$15.00 |
|                                               | Volume Purchases/Instructional | \$10.00 |
| SORT-1-Plus                                   | End User                       | \$10.00 |
|                                               | Volume Purchases/Instructional | \$ 7.00 |
| SourceTools                                   | End User                       | \$10.00 |
|                                               | Volume Purchases/Instructional | \$ 7.00 |
| Concurrent Programming Package                | End User                       | \$15.00 |
|                                               | Volume Purchases/Instructional | \$10.00 |
| Book                                          | Author(s)                      | Price   |
| Algorithms + Data Structures = Programs       | Wirth                          | \$27.95 |
| Elements of Programming Style                 | Kernigan, Plauser              | \$14.95 |
| Introduction to Pascal                        | Zaks                           | \$15.95 |
| Pascal User Manual and Report                 | Jensen, Wirth                  | \$ 9.50 |
| Programming in Pascal                         | Grogono                        | \$18.95 |
| Structured Programming                        | Dahl, Dijkstra, Hoare          | \$20.00 |
| Systematic Programming: An Introduction       | Wirth                          | \$26.00 |
| Tex and Metafont                              | Knuth                          | \$12.00 |
| Concurrent Euclid, The UNIX System, and Tunis | Holt, Graham, Lazowska, Scott  | \$15.95 |

Books and manuals are shipped FOB destination (within USA) or FOB Portland (outside USA).



# Pascal NEWSLETTER

NUMBER 7

OREGON SOFTWARE

SUMMER/FALL 1983

## UNIX compiler for 68000 released

At the USENIX Conference in Toronto, Oregon Software introduced the first high-performance Pascal compiler for the growing UNIX market on the Motorola MC68000. The compiler is available by itself or with the source-level, interactive Debugger and other software development utilities.

UNIX, developed by Bell Labs and licensed through Western Electric, is becoming a major development system on the 68000. Several dozen vendors are now offering UNIX or UNIX-like systems for applications development on the MC68000. Pascal-2 offers the benefits of Pascal without sacrificing the code quality required in the UNIX production environment.

Like other Pascal-2 products, the UNIX compiler performs eight optimizations on user programs, producing small, fast code. Oregon Software's benchmarks show that code produced by the Pascal-2 compiler is more compact and efficient than that of other high-level languages on 68000 UNIX systems, including C, and is an order of magnitude faster than interpretive or threaded languages.

### Debugger improved

A key feature of UNIX is the concept of supporting "tools" — standard, easy-to-use programs that aid a programmer in coding and development. Like Oregon Software's existing Pascal packages, Pascal-2 for UNIX contains a set of development tools — a debugger, execution profiler, program and text formatters, and cross referencers for program analysis.

As with the other Pascal-2 debuggers, the UNIX debugger allows the programmer to interactively solve logic errors in applications at the level of the source program, thus simplifying and speeding development. The UNIX debugger, however, runs as a separate process from the application code being debugged, keeps track of multiple compilation units, and performs breakpoint debugging without slowing down the application code.

The profiler and other utilities perform substantially the same as they do on other systems.

### Product unbundled

Pascal-2 for UNIX may be purchased as a complete package. Users also have the choice of purchasing the compiler only, and may opt for short-term or annual support.

Documentation includes the *Pascal-2 User Manual* with User's and Programmer's Guides, Debugger and Utilities Guides, and a detailed Language Specification; inserts for the UNIX Programmer's Reference Manual; and two textbooks on programming in Pascal.

## Version 2.1B release nears

Pascal-2 Version 2.1B is currently in field test. Release to general users is scheduled for Nov. 1. Numerous V2.1A bugs have been fixed and the ability to debug external procedures has been added. All supported users may request updated software and documentation from Oregon Software or their distributor.

## In this issue...

|                                          |         |
|------------------------------------------|---------|
| UNIX/68000 compiler released . . . . .   | Page 1  |
| RSTS SourceTools available . . . . .     | Page 2  |
| Pascal-2 on DEC Pro 350 . . . . .        | Page 2  |
| New releases for VAX, VERSAdos . . . . . | Page 3  |
| V2.1B ready . . . . .                    | Page 4  |
| Using unformatted files . . . . .        | Page 4  |
| Letters . . . . .                        | Page 6  |
| OPUS Communiqué . . . . .                | Page 7  |
| Accessing global symbols . . . . .       | Page 7  |
| Pascal-2's concurrency package . . . . . | Page 8  |
| Pascal-1 real-time facilities . . . . .  | Page 11 |
| The Log . . . . .                        | Page 18 |
| Errors, additions to manuals . . . . .   | Page 20 |
| Information exchange . . . . .           | Page 21 |
| Pascal standard . . . . .                | Page 21 |
| Sales staff . . . . .                    | Page 22 |
| Contest winners . . . . .                | Page 22 |
| Transition marked . . . . .              | Page 23 |



## **Source Tools also available for RSTS/E**

Oregon Software's SourceTools, a software management system, is now available for the RSTS/E operating system. The product was earlier released for the VAX/VMS and RSX-11M operating systems.

SourceTools allows programmers to manage access to sources and to document changes in sources over the development life cycle of a product.

In the past, software developers have relied on informal, manual methods to manage software development. With software projects growing in complexity and development teams increasing in size, these traditional methods of project management have eroded the real productivity of programmers.

SourceTools addresses this problem by providing an effective, automatic management scheme for software projects, particularly those involving numerous source files, joint development by several programmers, or the creation of cross-system software. SourceTools may be used with software developed in any computer language and with any standard text file.

The core of SourceTools is a package called SourceCon that controls the creation and modification of source files. Another program, MAKE, automatically keeps programs up to date as components change. Two other programs, TXTCOM and SEDIT, function together to ease the task of maintaining parallel sources on different systems.

### **Control of changes provides key**

Controlling access to a file, and recording any change to it, is the essence of a source-control system. With SourceCon, a file placed under source control becomes the base version. As the software is developed, each change to the base file is recorded separately, with a clear description of the person making the change, its purpose, the date and time, and the actual textual change.

Any version may then be reconstructed. Developers are able to generate releases from source-controlled files while simultaneously developing later versions of the same files. Developers are able to control the branching of software or documentation into several similar products without a proliferation of redundant files.

In addition, the clear audit trail helps developers isolate and correct human errors. In the case of large-scale mistakes, developers may replace the latest version of a file with an earlier one. Further, only one developer at a time has "write" access to source-controlled files, preventing conflicting changes from being made — a common cause of confusion and product delay in multi-programmer development.

### **'MAKE' rebuilds programs**

A second component of SourceTools is MAKE, a file rebuild-er that prevents the accidental inclusion of outdated modules into a large program compilation. After reading a user-written file describing the dependencies of each module on another, MAKE checks the date and time of each module and automatically rebuilds any that are out of date with respect to others. This facility improves program reliability and frees programmers for other tasks.

MAKE minimizes the number of commands required to rebuild programs and makes explicit the file dependencies.

### **Maintenance eased**

The third major portion of SourceTools consists of two programs that, used together, automate the maintenance of parallel source files. TXTCOM compares the contents of two files and generates a script of the differences between them; SEDIT, a stream editor, reads the TXTCOM edit script and applies the necessary changes to one of the parallel source files. Because only the script of differences has to be transferred, these programs minimize the time required for updating sources at remote sites or on different processors.

SourceTools is available from Oregon Software or its authorized distributors. Documentation includes a 75-page user manual.

## **Pascal-2 moves to DEC's Pro 350**

Beginning April 1, Oregon Software and authorized distributors are offering the Pascal-2 optimizing compiler for the DEC Professional 350, Digital's most advanced personal computer.

Developers may write Pascal-2 programs directly on the 350 under the RT-11 system. They also may use the 350 as a target machine, writing programs on the VAX or the PDP-11 and transferring the code to run on the 350.

The Pro 350 is a familiar environment for professional programmers. Many of them were trained on the 350's underlying PDP-11 hardware; Pascal-2 is the most widely used Pascal on the PDP-11. This means that a large body of applications programs may be moved to the 350 rapidly.



### Standard language allows integration

The Pascal-2 language implementation is standard across all Digital operating systems, plus UNIX. Pascal-2 for the Pro 350 supports all capabilities of standard Pascal, including conformant array parameters, and code developed under Pascal-2 may be moved to other operating systems that have standard Pascal compilers.

Pascal-2 offers low-level integration with the RT-11 system. Pascal-2 programs may call subroutines written in Pascal or assembler, allowing the user to take advantage of existing system software. Pascal-2's language extensions provide sophisticated I/O handling and access to system-specific operations.

### Tools included as options

Pascal-2 for the 350 includes, as an option, the full set of supporting tools available on other Pascal-2 systems: an interactive source-level debugger, an execution profiler, program and text formatters, and cross referencers for program analysis.

License fees for the complete Pascal-2/350 system are: \$450 for the compiler and support library, including 90 days' warranty, documentation, sample program, and subscription to our newsletter. A package containing the debugger and utilities is \$350. Software updates are \$300 per update. All prices are U.S.

Documentation includes the standard *Pascal-2 User Manual*.

## ***VAX cross-compiler targets any 68000 hardware, includes real-time capability***

Oregon Software's 32-bit, VAX-based Pascal-2 system for developing applications software to run on the Motorola MC68000 is now available.

The cross-development software allows users to target applications, including real-time programs, for virtually any 68000 hardware configuration. Most of the development effort takes place in Pascal, with development on the 68000 limited largely to final testing.

The cross-development system consists of the optimizing Pascal-2 cross-compiler, which supports the international standard; a library of routines allowing user programs to run without an operating system on the 68000 and providing the capability for concurrent programming; a cross-linker/assembler producing loadable code for the 68000; and program development utilities such as PASMAT and XREF.

The VAX/VMS cross-compiler supports 32-bit integers. A 16-bit version was released earlier for RSX on the PDP-11.

The Pascal-2 cross-compiler performs extensive error checking and uses the same optimization techniques as the other Pascal-2 compilers to produce small, fast code. Pascal-2 supports conformant array parameters, adhering to Level 1 of the international Pascal standard.

### Concurrency supported

Control of real-time processes is offered via the Concurrent Programming Package, which is an option to the system. The package enables users to develop all parts of an embedded or stand-alone system, even device drivers, in stan-

dard Pascal. The package offers true priority scheduling with a minimum of overhead.

The package consists of a library of procedures and functions that allow concurrent programming in the "process-monitor" style. This allows you to apply structured programming techniques to the problem of synchronizing the exchange of shared data among separate, concurrently executing processes.

The library also provides the interface with the hardware so that programs may run without an operating system. The library can be adapted to any hardware configuration; source code for all modules is provided to allow maximum flexibility when implementing the system.

### Available now

The development system is now available from Oregon Software or its authorized distributors; the exact price depends upon the options selected.

## **32-bit VERSAdos native announced**

In conjunction with release of the cross-compiler systems, Oregon Software also is announcing a native VERSAdos 32-bit compiler for the MC68000. This package includes an interactive source-level debugger, an execution profiler, and the other standard Pascal-2 utilities such as formatters and cross-referencers. The compiler generates code compatible with the Motorola linker/assembler and also has the Concurrent Programming Package as an option.

## How to read/write unformatted FORTRAN files from Pascal-2 under RSX-11M

by Joerg Donandt and Gerald Jahn

FORTRAN unformatted files are an efficient means of storing large data structures with respect to both storage and execution time. Unformatted data files require less storage space because redundancy is minimal. Conversions between internal binary form and a form readable by man (e.g., ASCII format) are unnecessary. For these reasons, we opted to use FORTRAN unformatted files to store digitized pictures, but we found that we needed to find a way to read and write them from Pascal programs — specifically those compiled with the Pascal-2 compiler. This article contains our solution and two samples of its use in Pascal programs.

### Unformatted file structure

The way data are stored and accessed in unformatted FORTRAN files causes the difficulty. Data are stored in variable-length records (132 bytes maximum length) that may wrap around block boundaries. Normally, carriage-control characters determine the way that the FORTRAN files are printed out, but the file management system cannot distinguish between a normal FORTRAN print file, generated using a `WRITE(6,100)...` statement where 100 is the format statement for the write, and an unformatted file generated by `WRITE(3)...` statement.

Access to these records depends upon a two-byte length counter, which isn't normally available to the Pascal programmer. When accessing these records, the 132 bytes are read into the run-time system's buffer as a "segment," and every segment begins with a two-byte segment descriptor that indicates the segment's place in the file by one of four values:

| Value | Description                                                      |
|-------|------------------------------------------------------------------|
| 1     | first segment of the file,                                       |
| 2     | last segment of the file,                                        |
| 3     | only segment of the file,                                        |
| 0     | any other segment between the first and last segments of a file. |

Every filled record holds 130 data bytes, 2 segment-descriptor bytes, and 2 length-counter bytes, for a total of 134 bytes per record.

In the 130 bytes following the segment descriptor, data are interpreted according to the variable-type and the sequence in which those variables were written into the file. For example, a FORTRAN expression such as `INTR*2-VALUE` is stored as two data bytes, with the byte representing the least significant digits first.

Multidimensional arrays, such as those in the sample programs, are stored in a "first-index-quickest" fashion: the elements of an array are stored in adjacent locations, the first index is incremented for each element stored until its range is exhausted, then the second index is incremented. Thus, a matrix `MAT(I,J)` is stored with the elements indices, as shown:

```
(1,1) (2,1) ... (I,1)
(1,2) (2,2) ... (I,2)
      :      :      :
      :      :      :
(1,J) (2,J) ... (I,J)
```

The storage required to hold a data item such as our picture-matrix (named `Pic` in the two sample procedures) would have to be declared `DIMENSION BYTE MAT(128,128)` consisting of  $128 * 128$  bytes = 16384 bytes, stored in 130-byte segments. The file consists of 126 filled segments and one 4-byte "appendix," for a total of 127 segments.

For such calculations, you need to be aware of the following numbers: every filled record holds 130 data-bytes, 2 segment-descriptor bytes, and 2 length-counter bytes, for a total of 134 bytes per record. The last record consists of 4 data-bytes and a 4-byte overhead, for a total of 8 bytes. The file's length is calculated as:

$$134 \text{ bytes/record} * 126 \text{ records} + 8 \text{ bytes} = 16982$$

These are stored in 33 decimal blocks (44 octal), with each block containing 512 bytes, leaving 4 bytes empty in the last block.

## Reading unformatted files

In order to read unformatted FORTRAN files, you must know **how** the unformatted file was written. With this in mind, look at sample procedure 1 (**Pic\_read**). The file was generated by the FORTRAN statements:

```
DIMENSION BYTE MAT(128,128)
:
WRITE(3) MAT
:
```

From the above we know that:

- The file holds only one variable: **MAT**.
- The variable is structured as a two-dimensional array, with indices ranging from 1 to 128.
- The elements of this array are byte-values: they require only one byte storage per element.
- The elements are stored in standard form as previously explained.

To express these storage and structure characteristics in Pascal, we define the segment as:

```
type
  Byte = 0..255;
  Pic = packed array[1..128,1..128] of byte;
```

Note that we use a packed array. Since **Byte** is a subrange of **integer**, each element of **Pic** would normally be allocated two bytes of storage because **integers** are stored in two bytes. The packed array, however, forces the compiler to allocate only the storage needed, in this case, one byte.

The procedure does three basic operations: opens the appropriate unformatted file; reads a record from it; and transfers the data bytes into the **Pic** matrix. The procedure repeats the loop of reading and transferring until the matrix is filled.

The **reset** statement must contain the following file attributes:

```
/VAR:132      Variable-length record (maximum 132 bytes).
/NOBLK       Records may wrap around block boundaries.
/FTN         Use FORTRAN's carriage-control convention.
```

In **Pic\_read** we have used the **read** statement for non-text files, violating the standard. The end-of-file (**eof**) does not work with non-text files, so we have introduced our own marker (**Endf**).

The statement **case** will handle the different possible segment-descriptors (**Code** in this procedure). The statement **code: for 3** would be somewhat inconsistent with our knowledge of **MAT**. Since **MAT** is large (about 16K Bytes), it must require more storage than a single record can provide. **Pic\_read** therefore aborts for **Codes** not in the range 0 to 2. The **FOR K:=1 TO 130** loop empties the **Data** section of the segment, byte by byte, and stores the data at the appropriate position (indexed by **I,J**) in the **Matrix**.

The procedure is straightforward, and, once you've understood the details, you can modify it easily to fit your needs.

A tricky approach, however, must be used in writing unformatted FORTRAN files from Pascal.

## Writing unformatted FORTRAN files

When we first tackled this problem, we had extreme difficulties writing segments into a variable-length record, non-text file. Indeed it was impossible. So we circumvented the problem with the following trick.

Instead of writing to a non-text file of variable-length records, we specify a normal text file and, using the **chr** function, we suppress the formatting normally done by the **write** statement. In addition, we used the **/FTN** attribute so that the FORTRAN run-time system (**FOROTS**) can find the attribute set when it tries to read unformatted data from the file.

Sample procedure 2 (**Pic\_write**) does not write the file in portions of "records" but byte by byte. It starts by writing the segment-descriptor as two bytes (low byte first). Then it fetches a byte from the matrix (indexed by **I,J**) and writes it to the file. When it has written 130 bytes, a **writeln** lets the system store the record away and start a new one. During the **writeln**, the file buffer is emptied and the length-counter (mentioned in the preceding discussion of file structure) is written to the file. Earlier, the length-counter could not be written, probably because the programmer is not allowed to use **writeln** statements on non-text files.

If you need to read/write unformatted FORTRAN files from the Pascal environment you ought to have a sound knowledge of the way both compilers treat your data. Although a general solution cannot be given, we hope that these remarks can help you solve your problem.

---

Mr. Donandt and Mr. Jahn are engaged in optical/digital image processing for pattern recognition purposes at The Institute of Applied Physics 1, University of Heidelberg, in Germany.

---

## Sample Procedure 1

```

procedure Pic_read(fname: string;
                  var matrix: pic);

type
  vlr = record { a FORTRAN variable-length record }
    code: integer;
    data: packed array [1..130] of byte;
  end;

var
  i, j, k: integer;
  datei: file of vlr;
  nxtrec: vlr;
  endf: boolean;

begin
  endf := false;
  reset(datei, fname.ch, '/var:132/noblk/ftn');
  while not endf do
    begin
      read(datei, nxtrec); { non-standard read }
      with nxtrec do
        begin
          case code of
            0: ;
            1:
              begin
                i := 1;
                j := 1;
                endf := false;
                end;
            2:
              endf := true;
              end; { abort, if another CODE occurred }
          for k := 1 to 130 do
            if j < 129 then
              begin
                matrix[i, j] := data[k];
                i := i + 1;
                if i > 128 then
                  begin
                    i := 1;
                    j := j + 1;
                  end;
                end;
              end;
            end { of with };
          end { of while };
        close(datei);
      end;
    end;
  end;

```

## Sample Procedure 2

```

procedure Pic_write(Fname: string;
                   var Matrix: pic);

var
  i, j, k: integer;
  datei: text;
  endf: boolean;
  c: byte;

begin
  endf := false;
  i := 1;
  j := 1;
  rewrite(datei, Fname.ch, '/var:132/noblk/si:-1/ftn');
  while not endf do
    begin
      c := 0;
      if (j > 127) and (i > 124) then
        begin
          endf := true;
          c := 2;
          end;
      if (j = 1) and (i = 1) then
        c := 1;
      write(datei, chr(c), chr(0)); {segment descriptor}
      for k := 1 to 130 do { fill a record into MATRIX }
        begin
          if j < 129 then
            begin
              write(datei, chr(matrix[i, j]));
              i := i + 1;
              if i > 128 then
                begin
                  i := 1;
                  j := j + 1;
                end;
              end;
            end;
          end;
          writeln(datei);
        end;
      close(datei);
    end;
  end;

```

## Letters

### To the Editor:

The following is a plea to whomever (or whatever) has control over the release of existing errors in the Pascal-2 compiler. I realize that there may be a bit of pride which must be swallowed to admit that there is a bug in one's software but it must be done. Anyone familiar with the complexities of a compiler, much less one that performs such significant optimization, realizes that such bugs must exist but should be able to accept that.

As a user of Pascal-2, I would have to say that finding a compiler error is very frustrating. It is not until I have spent several days tracing down the bug, only to find that

when I call one of your very helpful software support personnel that it is a bug which has been reported and known about, that I become very disappointed in the product. If I felt that I was the first person to encounter the error or had some chance to avoid it had I known about it, I would feel much more forgiving. (Obviously not admitting that it has been reported and resolved would be unforgivable.)

Of course we would all like every software product to be perfect but it isn't, not yet at least. So why kid ourselves? Let's communicate. The savings in real dollars as well as goodwill, I feel, would greatly outweigh the pride and postage it might cost you.

For instance, a compiler error could take as much as a day to track down to determine that it is in fact the



compiler and not a program error. If a programmer is paid nine dollars an hour and puts in an eight-hour day tracking the error down, it would be 72 dollars wasted. Multiply that by 100 installations which might come across the same error and already you have potentially wasted \$7,200. Don't forget to take into consideration the friction created between the EDP managers, their programmers and Oregon Software.

All this could have been avoided with a simple monthly letter to all supported sites which would describe the problem and the way, if any, to code around it. I realize that compiling such a letter may be non-trivial but its benefits out here in user land would outweigh that significantly.

One other request I have is for a fast single-pass version of the compiler with, perhaps, optional code generation. This again would create a great savings for us out here who must wait for 8 minutes, and sometimes more, for compiles

on our meager 11/34s only to discover a missing variable declaration.

Both of these requests I make with all sincerity and hope they will be considered with the same degree of seriousness.

Sincerely, Harry A. Levinson

### Editor's Reply:

Your plea for notice of known bugs has been heard, and we are in the process of improving our response. We are now better able to describe and track reported errors in a useful manner and hope to find a quick and painless way for users to keep up to date. We'll print a statement of the new policy in a future issue of the newsletter.

We have no plans for a single-pass version of the Pascal-2 compiler.

*BULL*

## OPUS Communiqué

### *Oregon Pascal Users Society*

Members should have begun receiving their membership list and member survey sheets on October 14, 1983 — just in time to contact others in the OPUS for Halloween.

The OPUS library will be opening soon. The logon for the OPUS library will be:

HELLO OPUS/LIBRT

or:

HELLO OPUS/LIBRSX

or:

HELLO OPUS/LIBRSTS

When you log on, a message appears:

For more information,  
type SYMSG.LST.

This file contains enough information to get you started on using the OPUS library. The OPUS library will be a read-only UFD. Dial-up lines are being installed now and the numbers should be in the October membership packet.

To put a program in the OPUS library, you must submit the source file only, with documentation of what it does, how to use it, how to compile and task-build/link it, and what its inputs and outputs are.

Everything submitted will be put into the library as long as it is consistent with OPUS goals and concepts. Send your routines to:

Bruce Williams,  
OPUS Coordinator  
c/o EOCOM  
15771 Red Hill Ave.  
Tustin, CA 92680  
(714) 730-5051, ext. 302

Some have expressed interest in having a DECUS-like meeting. Such a meeting is really time-consuming and requires a lot of effort. All that can be said at this time is maybe and only if there is more interest.

In the meantime, we will "nybble" on the membership list problem. More information will be in the next Communiqué.

## Accessing global symbols from Pascal programs

By Steve Poulsen

Many customers have asked how to access global symbols from a Pascal program. These are the global symbols used by the Linker and Task Builder, not the global-level variables defined at the start of a Pascal program. Users commonly want a data table defined in an assembler program from Pascal. It can be done, but it is tricky.

The concept of external data does not exist in Pascal, but the concept of external procedures does. The compiler generates a reference to the entry point of an external

procedure, using the first six characters in the procedure name as a global entry point. We use the compiler's ability to generate the address of a procedure to obtain the address of an external procedure.

When a procedure is passed as a parameter to another procedure, two parameters are passed. The first parameter is the static link used to access intermediate-level data in the proper context by the called procedure. Don't worry about it, we won't need to use it. The second parameter is the actual address of the procedure in memory. This is what we are after. Now all we need to do is to obtain the value of such a procedural parameter.

You must use an external procedure to obtain these two parts of a procedural parameter without violating Pascal's type checking rules because parameter typing cannot be performed across module boundaries. Define a module called GETADR.PAS (see example), a very simple function that takes two integer parameters and returns the second parameter as the function result. The trick is that we are really going to pass it a procedural parameter, and it will return the address of the entry point of the procedure. This function must be compiled as an external procedure to avoid the compiler's type checking rules.

The secret of accessing global symbols is to define the symbol as an external procedure and pass that procedure to a function that returns the address of the variable. For example, suppose that you want to access the variable \$DSW, which is the RSX directive status word whose location is defined by the Task Builder. You include the function

Getadr in a program as shown in the figure below. When the program is compiled and linked with the GETADR module (shown above the program), it prints the current value contained in the global variable \$DSW (usually a 1).

```
{ $nomain }
program Getadr;
{ Return the address of a procedure }

function Getadr(static_link, entry_point: integer): integer;
external;

function Getadr;
begin
  Getadr := entry_point;
end;

program Print_Dsw;

type
  Pointer = ^integer;

var
  Dsw_pointer: pointer; { Address of DSW }

procedure $Dsw;
external;

function Getadr(procedure Magic): integer;
external;

begin
  Dsw_pointer := Loophole(pointer, getadr($dsw));
  writeln(''$Dsw = ', Dsw_pointer);
end.
```

## Meeting design goals for Pascal-2 Concurrent Package

By Michael S. Ball

As computers increase in power and drop in price, many individual processors are being included as components of larger systems. These component processors are called embedded systems, and they are typically used for process control, data logging, communications processing, and signal processing. Such computers must deal with unique I/O devices and frequently must respond to external events in a short time. Traditionally, such applications have been coded in low-level languages, usually the assembly language for the machine. A part of the same tradition recognizes that such applications are difficult to code and even more difficult to debug. An efficient approach to concurrency is crucial to a successful embedded system.

### Concurrency models

Two basic models are used in writing concurrent programs. A "message model" looks at a concurrent program as a collection of processes connected by message queues. The processes communicate by passing messages between themselves. A "procedure model" sees a concurrent program as a group of processes that communicate through procedures and shared memory. The two views are in fact isomorphic, and any concurrent program written using one viewpoint can be rewritten using the other.

The difference appears to the programmer in the "primitives" supported by the systems. The primitives of a message-passing system are usually variations on two types: "send a message" and "receive a message." Special-purpose primitives may also exist, such as "send a message and wait for a reply," but the only required primitives are "send" and "receive." The primitives in a procedure-based system

are "acquire access to shared memory" and "release access to shared memory" (historically called "P" and "V"). Again, specialized versions may be implemented, but the only necessary ones are "acquire" and "release."

Each set of primitives may be implemented in terms of the other set and neither has any "special ability" not shared by the other. So how do we pick the approach to use? We look at the environment and applications intended for the system.

Let's consider two hypothetical examples. The first system consists of a number of microprocessors, each containing its own memory. Each microprocessor executes a single process, and the processes communicate over I/O channels. In this system, a message-passing scheme maps directly onto the hardware and is the obvious choice. The procedure-based system would have to simulate shared memory with messages, a profitless venture. In the second system, one or more processors share a common memory. Each processor may execute one or more processes, but large amounts of data are passed between processes. A procedure-based approach would be more appropriate in this system because communication takes place using shared memory anyway and a message-based system requires copying the data as it passes between processes.

An additional factor to consider is the type of I/O devices to be handled. The usual minicomputer or microcomputer device is easily treated as a process that shares memory with internal processes and communicates using signals. Devices with separate channel controllers may well fit more easily into a message model.

Our design goals matched better with the procedure model, so the Concurrent Programming Package supports that model of concurrent programming. The process-monitor approach of the Concurrent Package is based on a model that is slightly more refined than the basic one described above.

### Process-monitor model

Process-monitor style concurrent programming was initially proposed by C. A. R. Hoare [1979] and has been used in the programming languages Concurrent Pascal [Hansen 1977] and Modula [Wirth 1977].

The objective is to isolate all communication between processes into *monitors*. Each monitor consists of some shared memory (monitor variables) for communication and a set of procedures to manipulate this shared memory. Each queued procedure executes an "acquire access" operation on entry and a "release access" operation just before exit. All manipulations of the shared memory must take place within these monitor procedures, guaranteeing consistency of the data.

If a procedure within a monitor has to wait for some external condition, it does a "release access" operation for the monitor variables and waits — using a variation of "acquire access" — for some other process to make the condition true.

Although monitors need only "acquire" and "release," specialized forms of these operations improve efficiency and programming ease.

The Pascal-2 Concurrent Programming Package uses two special operations, **lock** and **unlock**, to control access to monitor variables. Each monitor has a special variable, a *gate*, that is used as an argument to **lock** and **unlock**. The monitor code calls **lock** on entry to the procedure and **unlock** on exit. Without a separate gate for each monitor, entry into any monitor would lock out entry into all other monitors.

The special operations **wait** and **signal** provide synchronization between processes. The process that executes **wait** unlocks the monitor variables and suspends execution until a **signal** is executed by some other process. **Wait** and **signal** operations are controlled by special variables called *events*. One speaks of "waiting for an event" or "signaling the occurrence of an event."

### Device interface

Embedded systems frequently have to drive a variety of I/O devices, and a useful concurrent programming system must provide support for device drivers. A device handler requires three things of the system:

- Access to the control registers for the device. On DEC's PDP-11 and Motorola's MC68000, device registers are mapped into the address space and manipulated like any other location in memory. The **origin** feature of the Pascal-2 compiler may be used to place a variable at the physical address of a device register; the register is then manipulated with normal assignment statements. Some additional procedures may be necessary for a computer that uses special I/O commands.
- Synchronization with the external device. Since external devices provide truly concurrent execution, even in a single-processor system, the hardware usually provides interrupts to signal that execution is completed. We can make the interrupt perform a "signal" operation as though it were an internal process.
- A way of providing mutual exclusion for access to the shared variables. The usual way of handling this in a computer is to turn interrupts off when manipulating data that is shared with the external device.

The Concurrent Programming Package meets the last two requirements with two new primitives. The first one is **makedevice**, which takes a gate as an argument and specifies



that the calling process is to execute with interrupts off. In addition, any process executing a **lock** on that gate has interrupts off until it executes a **wait** or an **unlock**. The process that calls **makedevice** is called a "device process."

Any device process can execute the **intwait** primitive, which has the same effect as a **wait** except that the signal is provided by an external device. During the time a process is waiting for an interrupt, other processes may execute within the monitor and have access to the shared memory.

The device process is sometimes called a "shadow process" for the external device. The combination can be considered a single process that executes on the external device during the **intwait** period and on the central processor at all other times.

### Implementation decisions

The simplest way to achieve concurrency on a single processor is the use of coroutines. The processes in a coroutine system execute one at a time, changing control only at specific points in the code. Using of the primitives described above, the control changes from one process to another only at calls to **wait** and **signal**. **Lock** and **unlock** primitives become unnecessary, as control changes only when the monitor gate would be unlocked anyway.

Device processes can be incorporated in a coroutine system quite simply. The **intwait** primitive saves and restores registers so that the execution of the device process has no direct effect on other processes. When the device process executes a signal that actuates an ordinary process, control does not pass directly to that process. Instead, the process is made ready and control passes to it at the next **wait** operation performed by an ordinary process.

The coroutine system is attractive in its simplicity. The overhead is minimal; a process swap costs little more than an ordinary procedure call. If the primitives are built into the compiler, the routine approach doesn't even need to save registers across a swap. On the other hand, response to I/O is delayed until a process voluntarily relinquishes control. A task with long periods of computation between interactions can slow response drastically.

A full concurrent programming scheme should also provide for process priorities and preemption of running processes. In such a system, process swapping occurs whenever a process with a higher priority than the current process becomes ready to execute. The disadvantage of such a system lies in the increased frequency of process swapping and the added complexity of each swap. Careful implementation can minimize the frequency of process swapping, however, and the added complexity is a small price to pay if you need to combine long computations with fast response.

### Our implementation

The Concurrent Programming Package provides Pascal-2 programmers with a set of primitives for programming embedded applications. The package supports multiple processes, device interfaces, and synchronization of processes. The entire application, from device drivers up, can be coded in standard Pascal.

The design goals for the package were to provide:

- Support for a variety of embedded systems. This includes a potentially large number of concurrent processes, some of which may have long sections of computation with interactions between other processes.
- Resultant software or programs able to run on a single mini or micro. Multiple processor systems are not difficult, but the coding details are dependent on the exact architecture of the system.
- Ability to write system-specific device handlers in Pascal.
- Minimal response time to external events and reduced cost of interprocess communication, so that it is cheap enough to be used whenever it fits the application.
- Resulting code capable of operating in a mixed ROM/RAM environment.
- A small, modular package. If an application does not require a particular capability, it should not have to pay the price for it.
- A self-instrumenting package that gives the programmer complete information on the state of the system when an error occurs.
- The ability to gather sufficient data on a program to spot and eliminate bottlenecks.

The Pascal-2 Concurrent Programming Package provides a scheme for multiple process priorities together with preemption. The extra cost is small compared to the capability gained. Process swapping is extremely efficient, with about 20 instructions executed in the worst case. This is little more than a simple procedure call.

The package includes a comprehensive set of diagnostic tools, including a process-by-process walkback in case of error. A log of the most recent primitive calls aids in the detection of subtle synchronization problems. The statistics gathered for each gate show the number of times the gate is used and the number of conflicts arising from two processes attempting to use the same monitor at the same time. Such statistics allow the detection of major performance bottlenecks.

The package meets all of the design goals and provides a useful addition to the software toolbox of system developers.



## References

C. A. R. Hoare, "Monitors: An Operating System Structuring Concept," *Communications of the ACM*, Vol 17, No. 10, October 1974.

Per Brinch Hansen, *The Architecture of Concurrent Programs*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1977.

N. Wirth, "Modula, A Language for Modular Multiprogramming," *Software Practice and Experience*, Vol 7, No. 1, 3-35 (1977).

---

Mr. Ball designed and implemented the Concurrent Programming Package at Oregon Software. Upon leaving Portland for a sunnier clime, he founded TauMetric Corporation, where he is a consultant for custom system software development. Address: 1094 Cudahy Place, Suite 302, San Diego, CA 92110, (619) 275-6381.

---

# Implementing real-time facilities in Pascal

by Hilding Elmquist and Sven Erik Mattsson

## Introduction

The increasing use of microcomputers in technical systems has made the art of real-time programming more important. We have previously used Concurrent Pascal [Brinch Hansen, 1971] in courses on real-time programming. However, we needed more flexibility. For example, we wanted to be able to demonstrate message-passing schemes and rendezvous. Furthermore, in order to be able to give the students a profound understanding of how concurrency is achieved, the nucleus or kernel of such a system has to be shown.

A natural way of introducing concurrency is to start with a sequential language such as Pascal and add necessary routines without changing the compiler or the support library. Per Brinch Hansen [1978] and Kriz and Sandmayr [1980] give descriptions of such routines in Pascal-like notations. However, it is not always possible to write these routines in Pascal because hardware facilities such as registers must be manipulated when the processor is switching between processes. Brinch Hansen [1978] translated the kernel by hand to assembly language.

The real-time kernel we've developed supports concurrent programming in Pascal, particularly in Pascal-1. The ker-

nel is written in Pascal, but it relies on a small number of assembly-language procedures for process creation, transfer between processes, and handling of interrupts. This set of routines is called the nucleus. The kernel implements semaphores for mutual exclusion and events for other synchronization. The kernel also provides the capability to program I/O handlers. With this real-time kernel, Pascal has the same expressive power as Concurrent Pascal. However, since the user's concurrent program is compiled with the standard Pascal compiler, there is much less security than in Concurrent Pascal.

In this article, we've described the implementation for Pascal-1 on the PDP-11 computer. The kernel has also been implemented on the Texas Instruments TMS 9900 and Motorola MC68000 processors.

## Kernel primitives

Three primitives handle creation, scheduling, and execution of processes. Others perform duties such as communication between processes and synchronization of their execution, timer control, and interrupt handling.

### Process handling

A process is declared as a parameterless procedure, which we refer to as "process-procedure" hereafter.

A process instance may be created by a call to **CreateProcess** from anyplace where the procedure could be called in the ordinary way:

```
procedure CreateProcess
  (procedure proced;
   memreq: unsignedinteger);
```

where *proced* is the process-procedure describing the process, and *memreq* is the memory requirement (in bytes) for the stack and the heap of the process.

The processes are scheduled according to their priorities. The priority of a process can be changed dynamically by calling:

```
procedure SetPriority
  (priority: integer);
```

where *priority* is the new priority of the process.

The main program must be converted to a process by a call to **InitKernel** before other processes can be created:

```
procedure InitKernel
  (memreq: unsignedinteger);
```

where *memreq* is the memory requirement (in bytes) for stack and heap (global variables excluded). The procedure **InitKernel** also creates a clock process and an idle process.

### Communication

Communication of data between processes is done with variables that are accessible to the process-procedures according to scope rules. The programmer must ensure mutual exclusion by the use of semaphores, as follows:

```
procedure InitSemaphore
  (var sem: semaphore;
   initval: integer);
```

```
procedure Wait
  (sem: semaphore);
```

```
procedure Signal
  (sem: semaphore);
```

where *sem* is the semaphore and *initval* is the initial value for the semaphore.

The synchronization between processes, such as waiting for a condition on a shared variable, is done by using the concept of an event, introduced by Brinch Hansen [1973]. There are three operations on events:

```
procedure InitEvent
  (var e: event;
   sem: semaphore);
```

```
procedure Await
  (e: event);
```

```
procedure Cause
  (e: event);
```

where *e* is the event variable, and *sem* is the associated semaphore for mutual exclusion.

An event must be initialized by a call to **InitEvent**, which associates the event with the semaphore for mutual exclusion. A call of **Await** delays the process and makes an implicit signal to the associated semaphore. A call to **Cause** by another process moves all delayed processes to the queue of the associated semaphore.

### Clock handling

The procedure **WaitTime** makes the calling process wait a specified time interval, which is expressed in ticks. A tick is 20 milliseconds.

```
procedure WaitTime
  (time: integer);
```

### Interrupt handling

The procedure **WaitIO** makes the calling process wait for a specified interrupt. The procedure sets the enable bit in the specified status register before suspending the running

process. Some other process is then scheduled for execution. When the interrupt occurs, the enable bit is reset.

```
procedure WaitIO
  (vecaddr: integer; var statusreg: integer);
```

The process waiting for the interrupt is indicated as ready for execution and the process having the highest priority is resumed. Only one process at a time can wait for a certain interrupt and this must be guaranteed by the user.

The PDP-11 has memory mapped I/O. Pascal-1 allows manipulation of the device buffers and status registers as ordinary variables by allowing specification of addresses in the variable declaration.

### Program organization

Since the programmer has to ensure mutual exclusion himself, it is important to organize the program in a way that aids in using the semaphores correctly. A natural solution is to collect the data, the semaphore for mutual exclusion, and the event variables in a record. Operations on the data are then conveniently done via a **with** statement. Ordinary Pascal allows recursion, thus procedures are reentrant, so it is possible to construct a set of procedures that operate on the shared data. This is the idea behind the monitor concept in Concurrent Pascal. Pascal-1 allows **type**, **variable**, and **procedure** declarations to be mixed, making it possible to collect the record declaration and the procedures together.

### Implementation

The introduction of concurrent processes means that code, processor and storage are shared resources. The problem of handling and protecting these resources must also be considered. Our aim is to use standard Pascal as far as possible. However, as hardware facilities such as registers must be manipulated, it is not possible to make the code completely portable. These computer-dependent parts are isolated in a small set of assembly procedures called the nucleus.

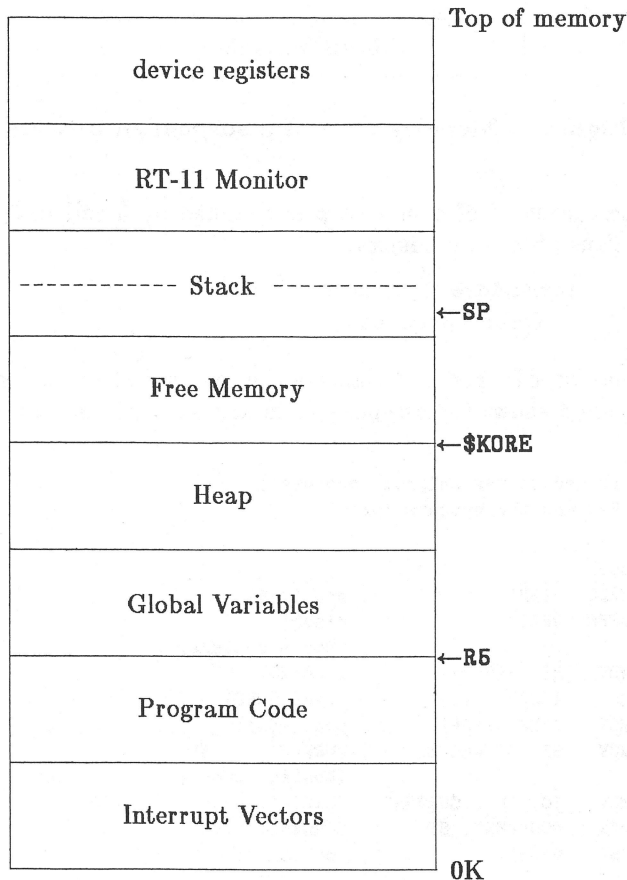
### Hardware

The PDP-11 computer has eight 16-bit general registers R0 through R7 [Digital Equipment Corporation, 1976]. The register R6 normally serves as the stack pointer (SP), pointing to the top element. The stack grows downward in the memory. Register R7 is the program counter (PC) of the processor.

### Pascal-1 run-time organization

Figure 1 shows the memory usage for a Pascal-1 program. All global variables are indexed relative to register R5. The

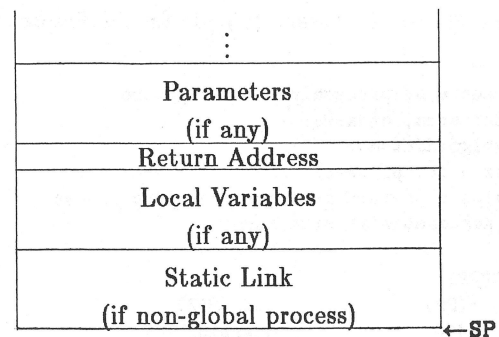
top-of-heap pointer is a global assembly variable called **\$KORE**. The stack and the heap grow toward each other. When a procedure is called or an allocation is made on the heap, a run-time check for overflow is performed.



**Figure 1. Pascal-1 program memory organization**

When a procedure is called in a Pascal-1 program, the parameters are calculated and pushed onto the stack. The first parameter is stacked first. When all parameters are stacked, a jump to the procedure is done with the assembler instruction **JSR PC, SUBR**, which pushes the return address (**PC**) onto the stack. The procedure then allocates space on the stack for the local variables.

All local variables and the parameters are accessed by indexing with **SP**. Intermediate-level variables are accessed by the use of the static links. The static link for an invocation of a procedure is a pointer to the stack frame of the latest invocation of the lexically enclosing procedure. To access an intermediate variable, it is necessary to follow the static links until the proper stack frame is reached and index by that base value. Information on the new static link is kept in **R4** during the subroutine jump. The static link is pushed onto the stack on top of the local variables (see Figure 2). The static link is not used in global procedures.



**Figure 2. The stack frame**

### Memory and process handling

A basic requirement for concurrent processes is that they have separate stacks for local variables and stack frames of called procedures.

The main program is converted to a process by a call of the nucleus procedure **InitNucleus**.

```

procedure InitNucleus
  (memreq: unsignedinteger;
   VAR startfree, endfree: unsignedinteger;
   VAR main: process);
  
```

where *memreq* is the memory requirement (in bytes) for main process, *startfree* is the returned start address for free memory, *endfree* is the returned end address for free memory, and *main* is the process variable for the main process.

An estimate of the memory requirements for stack and heap of the main process must be given as *memreq*. Global variables are excluded but space for stack frames of procedures called by the main process should be included. The procedure also returns information about free memory. The information is used later when subprocesses are allocated memory.

A new process is created by calling **NewProcess**, the nucleus procedure, as follows:

```

procedure NewProcess
  (procedure proced;
   startarea, endarea: unsignedinteger;
   var proc: process);
  
```

where *proced* is the process-procedure for the process, *startarea* is the start address for stack and heap area, *endarea* is the end address for stack and heap area, and *proc* is the process variable for the process.

Figure 3 shows the assembly code for `NewProcess`.

```
; procedure newprocess(procedure proced;
;   startarea, endarea:
;   unsignedinteger;
;   var proc: process);
; Creates a process from the procedure proced
; and associates it with 'proc'.

NEWPROCESS:
MFPS  -(SP)          ; push(PSW)
MRPS  #340           ; disable

MOV   8(SP), R0      ; R0:=endarea {child.SP}
MOV   8(SP), R1      ; R1:=startarea
MOV   R1, 04(SP)     ; proc:=R1
MOV   12(SP), R4     ; R4:=proced.staticlink

MOV   10(SP), -(R0)  ; pushchild(proced.startaddr);
MOV   R4, -(R0)      ; pushchild(R4)
MOV   R5, -(R0)      ; pushchild(R5)
MOV   R1, R2         ; R2:=R1
ADD   #2, R2         ; R2:=R2+2
MOV   R2, -(R0)      ; pushchild(R2) {$KORE}
MOV   #RESCHILD, -(R0) ; pushchild(RESCHILD)
MOV   R0, (R1)       ; proc:=R0 {child.SP}

MTPS  (SP)+          ; pop(PSW)
MOV   (SP), 10(SP)
ADD   #10, SP
RTS   PC

RESCHILD:
MOV   (SP)+, $KORE   ; {Start of child}
MOV   (SP)+, R5      ; pop($KORE)
MOV   (SP)+, R5      ; pop(R5)
MOV   (SP)+, R5      ; pop(R4)
MTPS  #0             ; enable
JSR   PC, 0(SP)+     ; proced
; {shouldn't come here}

LOOP:
JMP   LOOP           ; while true do;
```

Figure 3. Procedure `NewProcess`

Processes communicate by using variables that are accessible for the process-procedures according to the ordinary Pascal scope rules. There is no problem using global variables since they are all accessed relative to register `R5`. The parameter *proced* contains information about the static link and the address of the code for the procedure.

A process is suspended by a call to the nucleus procedure `Resume` or `IOresume` or by an interrupt. The context (relevant registers, `$KORE`) of the process is stored on the stack (see below) of the process. The stack pointer is saved just below the heap of the process. The address to this location is the value of the process variable. The memory area of a suspended process is shown in Figure 4. The nucleus has a copy of the process variable of the currently running process in an assembly variable named `CURRENT`.

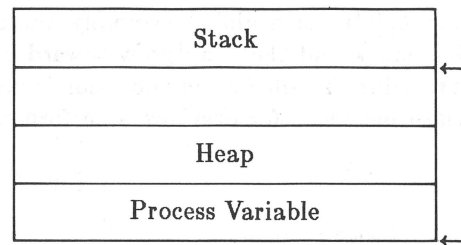


Figure 4. Memory area of a suspended process

The execution of a process *p* is resumed by a call to the nucleus procedure `Resume`:

```
procedure Resume
(proc: process);
```

where *proc* is process variable of the process to be resumed. Figure 5 shows the assembly code activated by the call.

```
; procedure resume(proc: process);
; Resumes the process 'proc'.

RESUME:
MFPS  -(SP)          ; push(PSW)
MTPS  #340           ; disable
; {Store context}
MOV   R5, -(SP)      ; push(R5)
MOV   $KORE, -(SP)   ; push($KORE)
MOV   #RES, -(SP)    ; push(RES)
MOV   SP, 0CURRENT   ; CURRENT := SP
; {Resume 'proc'}
MOV   10(SP), CURRENT ; CURRENT:=proc
MOV   0CURRENT, SP   ; CURRENT:=proc
JMP   0(SP)+         ; switch

RES:
; {Restore context}
MOV   (SP)+, $KORE   ; pop($KORE)
MOV   (SP)+, R5      ; pop(R5)
MTPS  (SP)+          ; pop(PSW)

MOV   (SP), 2(SP)
ADD   #2, SP
RTS   PC
```

Figure 5. Procedure `Resume`

The nucleus procedure `IOresume` makes the current process wait for a specified interrupt and resumes another process:

```
procedure IOresume
(proc: process;
 vecaddr: integer);
```

where *proc* is the process variable of the process to be resumed, and *vecaddr* is the vector address for awaited interrupt. When the interrupt occurs, the current process, which might not be *proc*, is suspended and the process waiting for the interrupt is resumed.



The LSI-11 has only two priority levels: interrupts enabled or disabled. The nucleus contains two procedures that change the status of the processor.

```
procedure enableinterrupts;
```

```
procedure disableinterrupts;
```

All processes, including the main program, start with the interrupts enabled.

Upon interrupt, the content of PC is automatically pushed onto the stack. PC is loaded from a preassigned memory location called an *interrupt vector*. The actual location is chosen by the device interface designer and is located in low memory address. When the specified interrupt occurs, the suspended process should be resumed. The desired effect of the interrupt is thus that a call would be done to a procedure:

```
IntResume(iosuspended: process);
```

Procedure **IntResume** is similar to **Resume**. The argument **iosuspended** is the process that called **IOresume** with the vector address corresponding to the interrupt. However, the hardware handling of the interrupts on a PDP-11 allows only the calling of a parameterless procedure when the interrupt occurs. This hardware facility is not convenient in connection with reentrant routines that are called by several processes.

One solution to this problem is to dynamically create, for each process, a small procedure that calls **IntResume** with the appropriate argument. In fact, the only thing that needs to be done is to place the assembler instruction **JSR PC,INTRESUME** above the top of the stack of the process calling **IOresume** and the address to this instruction in location **vecaddr**. When the interrupt occurs and the **JSR** instruction is executed, the return address is stacked on the stack of the currently executing processes. However, the return address is exactly the stack pointer (i.e. the value of the process variable) for the I/O-suspended process. When the interrupt occurs, the process that should be resumed is made known to the interrupt handling routine in an efficient manner.

### Processor management

The kernel primitives contain switches of the processor between the processes. The nucleus has procedures that can suspend and resume processes, but the kernel must decide which processes should be running. The kernel stores relevant information about the processes in records of type **processrec** (see Figure 6). The kernel keeps these records in different double-linked lists. Every list has a head of the same type as the rest of the elements in the list in order to avoid special handling of empty lists.

The processes that are competing for the processor are kept in **readyqueue**, and the variable **running** keeps track of the running process.

One list of process records is associated with each semaphore **waiting** and each event **delayed**. All processes waiting a specified time are kept in a single list **timequeue**. They are ordered according to the increasing waiting times. The process record contains one field **time**, which contains the waiting time relative to the preceding process in the time queue. The waiting time for the first process is relative to the current time. This is an efficient way of organizing the time queue because the clock process needs only to decrement the time field of the first process at each clock tick. The relative waiting times are calculated by the procedure **WaitTime**.

Only one process is allowed to wait on a specific interrupt. Therefore, no list of process records is needed. A reference to the process record of the waiting process is kept in a local variable in each instance of the procedure **WaitIO**.

The scheduling rules make it natural to order the elements in the ready queue and in the queues associated with semaphores according to their priorities. The order is unimportant in a queue associated with an event, so its elements can be inserted at the end.

### Kernel structure

The data structure of the kernel is a shared resource and mutual exclusion is accomplished at this level by disabling interrupts.

The entry routines of the kernel must be visible from the user's program. The Pascal-1 compiler allows separately compiled modules with procedures and functions. A global routine in a separately compiled module is visible from other modules, if the compiler's external module switch is turned on (**(\$E+)**). The user who wants to use a separately compiled routine must define a procedure heading followed by the keyword **external**. A file that contains predefined types and procedures declared external is available as a prefix to the user programs.

### Conclusion

These techniques allow programmers to introduce concurrency in ordinary Pascal without changing the compiler or the support library, an approach suitable for educational purposes. The kernel is used in an undergraduate course in real-time programming. Programs for control of a physical process, operator communication and point-to-point protocol for computer communication have been implemented by the students. It is also possible to use the nucleus to test new primitives for concurrent programming. Routines for message passing and for the rendezvous concept of Ada have been implemented.

The resulting kernel is comparable to Concurrent Pascal or to Texas Instruments  $\mu$ P (Microprocessor Pascal). Hoppe [1980] presents a similar kernel with message-passing mechanisms written in Modula-2 [Wirth, 1980]. Some extensions have been made to the kernel in order to make it easier to work with. We've used a D/A converter, which outputs an analog signal that is displayed on an oscilloscope, to show what process is currently running. It is also possible to take a snapshot and generate a status report for the processes. A multi-terminal handler has been written in Pascal. It has been connected to the `read/write` statements of Pascal by handling the software interrupts generated by the support library.

The nucleus (Figure 6) shows what has to be added in order to obtain concurrency in Pascal. The code is written in a straightforward way to make it easy to understand. The primitives introduced in the nucleus are similar to those of Modula-2. Interrupt handling is included in an efficient way.

## References

- P. Brinch Hansen, *Operating System Principles* (Englewood Cliffs, New Jersey: Prentice Hall Inc., 1973).
- P. Brinch Hansen, *The Architecture of Concurrent Programs* (Englewood Cliffs, New Jersey: Prentice Hall Inc., 1978).
- Microcomputer Handbook* (USA: Digital Equipment Corporation, 1976).
- H. Elmqvist and S. E. Mattsson, "Implementation of Basic Primitives for Concurrent Programming in Pascal," (Lund, Sweden: Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7230)/1-023/, 1981[a]).
- H. Elmqvist and S. E. Mattsson, *A Real-Time Kernel for Pascal* (Lund, Sweden: Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7231)/1-023/, 1981[b]).
- J. Hoppe, "A Simple Nucleus Written in Modula-2: A Case Study," *Software Practice and Experience* 10 (1980): 697-706.
- J. Kriz and H. Sandmayr "Extension of Pascal By Coroutines and Its Application To Quasi-Parallel Programming And Simulation," *Software Practice and Experience* 10 (1980): 773-789.
- Pascal-1, Version 1.1 for RT-11* (Portland, Oregon: Oregon Software Inc., 1978).
- N. Wirth, *Modula-2* (Zurich, Switzerland: Institut fur Informatik, ETH, 1980).

Professors Elmqvist and Mattsson teach real-time programming in the Department of Automatic Control at the Lund Institute of Technology in Sweden. They have designed a kernel that supports concurrent programming in Pascal-1. Their approach shows what must be added to set concurrency in a sequential language and it is therefore suitable for education. This description of their real-time kernel was presented at the IFAC Symposium on Software for Computer Control 1982 and is reprinted here by permission of the International Federation of Automatic Control. The authors want to thank Leif Andersson for many good ideas and stimulating discussions.

**{Real-Time Kernel for Pascal.}**  
**{Use the nucleus prefix.}**

CONST maxpriority = 1000;

```
TYPE unsignedinteger = 0..65535;
processref = ^processrec;
semaphore = ^semaphorerec;
event = ^eventrec;
processrec = RECORD
succ, pred: processref;
proc: process;
priority: integer;
time: integer;
END;
semaphorerec = RECORD
counter: integer;
waiting: processref;
END;
eventrec = RECORD
reentry: semaphore;
delayed: processref;
END;
VAR running, readyqueue, timequeue: processref;
freetop, freebase: unsignedinteger;
```

```
PROCEDURE put(p, q: processref);
{Inserts process record p before process record q in q's list.}
BEGIN
p^.succ := q;
p^.pred := q^.pred;
q^.pred^.succ := p;
q^.pred := p;
END;
```

```
PROCEDURE remove(p: processref);
{Removes processrecord p from its list.}
BEGIN
WITH p^DO
BEGIN
pred^.succ := succ;
succ^.pred := pred;
END;
END;
```

```
PROCEDURE putpriority(p, q: processref);
{Inserts processrecord p in queue q according to priority.}
VAR p1: processref; pri: integer;
BEGIN
pri := p^.priority;
p1 := q^.succ;
WHILE (p1 <> q) AND (pri >= p1^.priority)
DO p1 := p1^.succ;
PUT (p, p1);
END;
```

```

PROCEDURE schedule;
BEGIN
  IF readyqueue^.succ <> running THEN
    BEGIN
      running := readyqueue^.succ;
      resume(running^.proc);
    END;
END;

{$E+} {external}
PROCEDURE createprocess(PROCEDURE proced; memreq: unsignedinteger);
  VAR child: processref;
  BEGIN
    disableinterrupts;
    freetop := freetop - memreq;
    new(child);
    child^.priority := 1; {Default priority.}
    putpriority(child, readyqueue);
    newprocess(proced, freetop, freetop+memreq, child^.proc);
    schedule;
    enableinterrupts;
  END;

PROCEDURE initkernel(memreq: unsignedinteger);
  CONST clockarea = 100; idlearea = 100;
  BEGIN
    disableinterrupts;
    {Create readyqueue with running.}
    new(running);
    new(readyqueue);
    readyqueue^.succ := running;
    readyqueue^.pred := running;
    running^.succ := readyqueue;
    running^.pred := readyqueue;

    {Create empty time queue.}
    new(timequeue);
    timequeue^.succ := timequeue;
    timequeue^.pred := timequeue;

    running^.priority := 1;
    initnucleus(memreq, freebase, freetop, running^.proc);

    createprocess(clock, clockarea);
    createprocess(idleproc, idlearea);
  END;

```

```

PROCEDURE initsem(var sem: semaphore; initval: integer);
  BEGIN
    new(sem);
    WITH sem^ DO
      BEGIN
        counter := initval;
        new(waiting); {Empty waiting queue.}
        waiting.succ := waiting;
        waiting.pred := waiting;
      END;
    END;

PROCEDURE wait(sem: semaphore);
  BEGIN
    disableinterrupts;
    WITH sem^ DO
      BEGIN
        IF counter > 0 THEN
          counter := counter - 1
        ELSE
          BEGIN
            remove(running);
            putpriority(running, waiting);
            schedule
          END;
        END;
      END;
    enableinterrupts;
  END;

PROCEDURE signal(sem: semaphore);
  VAR p: processref;
  BEGIN
    disableinterrupts;
    WITH sem DO
      BEGIN
        IF waiting <> waiting^.succ THEN
          BEGIN
            p := waiting^.succ;
            remove(p);
            putpriority(p, readyqueue);
            schedule
          END
        ELSE
          counter := counter + 1
        END;
      END;
    enableinterrupts;
  END;

```

**Figure 6.** This listing contains some parts of the kernel. The procedures **SetPriority**, **Initevent**, **Await**, **Cause**, **WaitIO** and **WaitTime** and the processes **Idleproc** and **Clock** are not included.

# The Log: Pascal-1 and Pascal-2

Oregon Software's previous Pascal Newsletter described the significant changes in Pascal-2 V2.1A. This log details bugs fixed in Pascal-2 V2.1B, which may now be ordered. Several fixes listed for V2.1 involve the support library; such fixes also apply to Pascal-1.

The changes described apply to all versions of Pascal-1 or Pascal-2 unless a specific operating system is specified. When possible, work-arounds are given for V2.1A users.

## Pascal-2

Version 2.1B corrects these problems for all PDP-11 systems.

### Debugger altered registers

The Debugger altered some of the program's registers when the C, S, or P commands were used to start a program running under the Debugger. Work-around: use G to begin program execution. After the program has started executing, the C, S, and P commands work correctly.

### Negative zero

The write procedure sometimes printed a negative sign for negative real numbers that round to zero. For example, `write(-0.010:7:1)` printed the value `-0.0`. The correct result is `0.0`. This error did not occur with all such numbers.

### 'Sin' and 'Cos' problem

The double-precision `sin` and `cos` routines did not return the correct value when their argument was exactly `0.0`. Work-around: use a special check for the argument `0.0`, as in the following example:

```
function Xcos(X:real):real;
begin
  if X = 0.0 then Xcos := 1
  else Xcos := Cos(x)
end;
```

### Line numbers thrown off

The use of included files threw off error walkback line numbers. Work-around: use the new syntax, with quotes around the file name.

```
%include 'filename';
```

The old syntax may not continue to be valid in the future, anyway.

### Debugger changed variables

When a fatal error is detected in a user's program, the Debugger regains control after the error message is printed. However, the Debugger did not print the correct values for some variables, because the error-reporting procedures in the support library did not preserve registers and the Debugger cannot print the values of variables stored in those registers. V2.1B fixes the problem for many run-time errors, but certain kinds of run-time errors may still change the values of some variables. Please be aware of this restriction: **When a run-time error terminates a program, some variable values printed by the Debugger may be inaccurate.**

### Dynamic allocation incorrect

When a 2-byte block was allocated from a 4-byte free block in memory, the remaining 2 bytes in the free block were not handled correctly during execution. The problem caused memory fragmentation or even odd address traps when the program ran.

### Multiple errors on file output

The error message mechanism attempts to write any partial output line to the output file as part of the `close` operation after a fatal error. If the error was detected during a write to an output file, such as a disk, the attempt triggered the "multiple errors detected" message instead of the actual problem. In V2.1B, the support library marks the second occurrence of an I/O error for file output as "non-fatal." This should prevent the program from aborting with the "multiple errors detected" message.

### Miscellaneous

Incorrect results were obtained from functions that returned structured-type values.

Run-time traps and incorrect calculations occurred when operations were performed on packed structures.

Spurious compile-time errors were reported when conformant array parameters were used with nested procedures.

## Changes to RSX

Version 2.1B for RSX corrected these problems:

### 'Getpos' returned incorrect location

The `getpos()` procedure did not always return the correct location of the next record about to be read when the input file was a `text` file.

### 'Put' didn't always put

When a terminal, line printer, or paper tape punch was opened as a file type other than `text`, data was not correctly written to the device when a `put()` was used. For



example, if a terminal was opened as a **file of integer**, and **put()** statements used to write data to the terminal (two characters per integer), nothing was printed on the terminal.

#### 'Ioerror' always 'true'

If **ioerror** was the first operation performed on a **text** file, an interaction with Lazy I/O caused the return of the value **true** even when no error existed. Work-around: use the fourth parameter of the **reset** or **rewrite** call to determine whether the file is correctly opened. The value of the fourth parameter is -1 if the file cannot be opened.

#### Errors not trapped

The run-time errors "File is not a random access file" and "Seek() to record zero" were not correctly trapped when **noioerror()** was used with a file.

#### Profiler overlay omitted

The overlay description file **PAS.ODL** for V2.1A did not explain how to overlay a Pascal-2 program compiled with the **/PROFILE** option. For V2.1B, **PAS.ODL** properly describes overlay procedures for programs that use the Profiler.

#### 'Not enough memory'

When a Pascal task is initialized and the section **\$\$HEAP** has not been expanded, the support library attempts to allocate 2K words for the stack by extending the task. This works fine until the size of the task reaches 30K words. At this point, less than 2K words remain for the stack and the compiler reports "not enough memory." In 2.1B, Pascal-2 uses any remaining space for the stack when a task exceeds 30K words, allowing users to write larger tasks before using overlays.

#### Incorrect file size returned

When the fourth parameter is used with the **reset** procedure, Pascal-2 should return the size of the file, in blocks. However, Pascal-2 was returning a file size of one block for an empty file. The file size for an empty file is now reported as zero.

#### /APD/RW causes lost line

A line of information may be lost under the following conditions: an existing **text** file is opened via a **reset** with the **/APD/RW** switches to permit appending data to the file, and a **write** statement is used without a **writeln** to write the characters to the file, and the file is then closed. The problem is caused by the use of **reset** to open the file; **reset** normally opens files for input only. When the **/APD/RW** switches are used, the file is really an output file, so any partial lines should be flushed out when the file is closed. Use **writeln** to be sure that the last line is written to the file before the file is closed.

#### 'Sayerr' incorrect

The **sayerr** routine did not correctly print the text for I/O error when the RSX I/O error codes and directive error codes (stored in a byte) were ambiguous. In version 2.1B, the directive error codes are biased by 128 to distinguish them from I/O error codes. V2.1A users should be aware that errors such as "Illegal user buffer" might be reported when "Device driver not resident" is actually the error.

#### Record lengths for 'text' files

When **text** files containing lines longer than 132 characters were opened with a **reset** statement, a "record length error" could be generated because Pascal allocates a maximum buffer size of 132 characters. Work-around: use the I/O control switch **/VAR:nnn** where **nnn** is the maximum line length of the data in the file. The value **nnn** can be greater than 132.

#### 'Rename' didn't

When an explicit version number was given for a file to be renamed and that file was not the most current version, the **rename** operation completed as expected, but the directory entry for the most current version of the file was removed instead of the entry for the file that was renamed. V2.1A users should not give explicit version numbers for files being renamed.

#### Character dropped on line wraps

When a line longer than 132 characters is written to a **text** file, the line wraps and the line is broken at 132 characters as if by a **writeln** statement. Under V2.1A, a character was dropped when the line is wrapped this way on disk files. Terminal output wraps correctly. Work-around: use a **writeln** statement before outputting more than 132 characters.

#### Message vague

The run-time error message "multiple errors detected" does not provide any information about the actual errors detected in the program. The error message is printed when a run-time error occurs while an error message is being printed, and it usually indicates a severe error, such as writing over support library code. In version 2.1B, the support library causes a run-time trap so that RSX prints the contents of the hardware registers as follows:

|    |                               |
|----|-------------------------------|
| R0 | User PC of original error     |
| R1 | Error code for original error |
| R2 | Secondary error PC            |
| R3 | Secondary error code          |
| R4 | I/O status if I/O error       |

The error codes are in Appendix B: Run-Time Error Messages of the Programmer's Guide.

# Errors, additions to manuals

## Sample program errs

The sample program REVERSE listed in the *Pascal-2 User Manual* for V2.1 does not operate properly. The second line of the program being reversed is not printed because of an interaction with lazy I/O and the way the REVERSE program is written. To operate correctly, a boolean variable called **Done** must be inserted into the program. The main loop in the program must be modified as shown:

```
repeat
  new(x);
  with x^ do Getpos(f, block, offset);
  x^.next := p;
  p := x;
  Done := eof(f);
  if not Done then readln(f);
until Done;
```

## Changes to RSTS

Version 2.1B for RSTS corrects these problems:

### No record-oriented I/O under RSX

When the RSX version of Pascal is run under the RSX emulator on RSTS, it is not possible to perform I/O on record-oriented devices other than the user's terminal. The RSTS system returns an error status of -37 for all such I/O requests because the operating system does not properly simulate RSX I/O to record devices such as the line printer. I/O to the user's terminal is handled correctly. In V2.1B, code has been added to enable the support library to recognize RSTS emulation and perform I/O to record devices correctly.

### 'Read' may hang

A **read** performed on an integer or real hangs forever if the user types ^Z with I/O error trapping turned on and without entering a valid number first.

### Second real value lost

The value of the second real variable is lost in programs that attempt to read two reals in succession, but only when the second real contains fewer characters than the first. Workaround: add a leading 0 to the second real.

### Iostatus printed incorrect values

**Iostatus** produced the wrong values when an I/O error occurred, causing the error routine to print either a zero or a large negative number.

### Modules missing in /eis version

The EIS version of the support library for V2.1A did not contain modules for the exponent function and the natural logarithm.

### Miscellaneous

**LIBDEF.PAS** was not properly reflecting the data stored in the file variable. This resulted in **FDUMP** not giving proper results as well as prohibiting the user from testing any of the device status conditions accurately.

**OPFDMP.PAS** has been modified to properly print the device number if there is one.

**OPERRO.PAS** now prints the unit number of the device if an I/O error occurred.

The entry point for **P\$DISP** was in the user library when it should have been in the run-time system. Attempts to run the examples in the manual that accessed this entry would fail.

## Changes to RT-11

### Error reporting improved

The **fatal** error status at location 53 octal was not being properly set or tested by the support library. This resulted in some command files not terminating when a fatal error was encountered.

In conjunction with the error status problem, the low-level routines (**.READ/WRITE**) would also set the error condition bit if a fatal transfer error occurred. This would cause Pascal to print the "multiple errors" message.

The **Sayerr** routine was added to print the error text of low-level RT-11 system calls.

**Iostatus** would produce the wrong values when an I/O error occurred, causing the error routine to print either a zero or a large negative number. **Iostatus** has been modified to return a negative value if a system error occurs on a file, a zero if no error occurs, and a positive number if the problem is a support library error. The error number is returned as a full 16-bit integer value.

The support library now saves the error condition code (if any) from the last **reset** or **rewrite** operation so that the user may call **Ioerror** and **Iostatus** to determine a possible cause if the operation fails. This will work properly only if called before any other I/O operation is performed.

The reason is that the saved location is also set by the .READ/WRITE code since **reset** could imply a .READ, which could cause the failure.

### 'Read' may hang on reals

A **read** performed on an integer or real hangs forever if the user types ^Z with I/O error trapping turned on and without entering a valid number first.

### Second real value lost

The value of the second real variable is lost in programs that attempt to read two reals in succession, but only when the second real contains fewer characters than the first. Workaround: add a leading 0 to the second real.

## Pascal has a standard

Pascal now officially has an international standard. The International Standards Organization, in a vote tallied this summer, adopted a standard language definition for Pascal. The action followed earlier adoption of an American standard that is a subset of the international one, lacking only conformant array parameters.

The sequence leading to adoption of the Pascal standard began in December 1982, when ANSI and IEEE agreed on the American standard, identical to the international draft standard except for conformant array parameters. At the same time, the Joint Pascal Committee of ANSI and IEEE recommended adoption of the international standard Level 1 (including conformant array parameters) to the U.S. committee known as X3J9, which then voted "yes" on the international standard at the next meeting. Previously, the U.S. was one of three "no" votes. This time, the ISO standard passed with no dissenting votes and one abstention.

## Work continues on extensions

The Joint Pascal Committee of ANSI and IEEE continues to work on a "candidate extension library" for extensions that will resolve the known weaknesses of standard Pascal. The ANSI-IEEE committee has agreed on a number of minor issues but has not resolved differences over major issues. Proposals are circulating on these issues, which remain "work in progress." A report is due in December.

Copies of the international standard are available from:

Document number BS 6192:1982.  
British Standards Institute  
Marylands Avenue  
Hemel Hempstead  
Herts HP 2 4SQ

## Information exchange

If you need information on technical applications involving Pascal, or if you have an application that might interest other users, send us a brief description for inclusion in the "Information Exchange." Your description should follow the format of the items below. Interested parties may contact one another directly.

**Log Management System and RMS Interface**, for RSX-11M; a menu-driven 50-program package that incorporates a custom-developed interface (available separately) between Digital's Record Management Services (RMS) and Pascal-2 V2.OK. The log package supports a multiplicity of data reporting features with subsystems for accounting, timber sales, and sales data accumulation. The RMS interface gives the user access to all of the RMS sequential, random, and multi-keyed capabilities. Contact: Jim Bombardier, Wasser and Winters Co., P.O. Box 396, Longview, WA 98632, (206) 423-1080.

**DCL Command PASCAL**, for the Oregon Software Pascal-2 V2.1A compiler; software package contains patches and files to enable a new DCL command **PASCAL** and to make **HELP PASCAL** work. All files are for RSX-11M V4.0 only. For information, contact: Mr. Bruce Williams, OPUS Coordinator, c/o EOCOM, 15771 Red Hill Avenue, Tustin, California 92680.

## Distributors attend annual workshop

Oregon Software Distributors from Europe, Canada, Japan and Australia received technical information from Oregon Software development teams and exchanged viewpoints on the needs of customers at their second annual workshop.

The two days of discussions in Portland followed DEXPO East and preceded Oregon Software's sixth Annual Open House. The first day was devoted to technical presentations and a hands-on session, where distributors ran concurrent-programmed games on the 68000, used SourceTools in a simulated development environment, and compiled programs under UNIX on the PDP-11. The second day was devoted to round-table discussions in which participants exchanged views on the needs of customers, the quality and timeliness of written materials, and the need for additional information.

## Customer contact is the key for sales staff

In the last issue, we introduced our marketing director, David Cloutier, and manager for general distributors, Pat Rau. We'd also like you to know our sales staff, who they are and what they do.

Lorie, Terry, and Tim handle customer support and sales. Their jobs require them to spend lots of time on the phone, providing product information, following up on sales, and responding to customers' requests for help. They respond to inquiries from prospective customers, which come in response to advertising, articles, or as referrals from our friends. The sales staff provides prospects and current customers alike with additional product information.

Their jobs don't end with a sale. They rectify any shipping problems and keep their accounts in support.

### Lorie Griffith

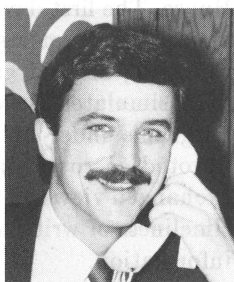
Lorie, our senior sales person, has a strong science background, including some contact with computers as a chemistry major in college and as an assayist on her first job. Later, she became a computer operator for a sawblade manufacturer. Before she joined Oregon Software, Lorie did customer support for an applications software company. Her major project was a large automated inventory and accounting system for an automobile importer. As part of an 18-month contract, she did some programming and wrote user documentation in addition to providing support.



Lorie lives in rural Washington, where she raises livestock and is active in community projects.

### Tim McMenamin

Tim says that "people make the job worthwhile"; he likes the service to customers as much as the software itself. He enjoys negotiating with distributors and finds the contact with large firms stimulating.



A native of Wisconsin, Tim came to Oregon for skiing and schooling.

A journalism and political science major at college, he worked on the staff of a state senator and a congressman. Before coming to Oregon Software, Tim sold DBMS applications and CP/M program generators for a small software house. He likes the technical challenge of sales and is taking computer science courses to build his program-

ming knowledge.

When he's not at work or school, Tim enjoys all forms of skiing, camps out in the local mountains, and runs for exercise.

### Terry Juve



Terry is a recent addition to the staff. Trained as a programmer, she likes to work with other programmers. She also enjoys direct contact with the customer.

Terry recently earned an associate degree in applications programming at Portland Community College, and she's starting on a bachelor's degree in computer science at Portland State University.

Her sales background has been in commercial casualty insurance for various brokerage houses in Portland, and she's taken business courses as well.

Between work and school, she likes to spend the little bit of free time she has reading science fiction and psychology; she likes to cook, hike, dance, and play an occasional softball game.

## Bug Contest winners crowned

After some tough decisions, we have the winners for the Bug Contest (see Newsletter No. 4). We couldn't decide on a "best" bug so we have co-winners. The prize category itself forced us to make some hard choices, but we finally decided on an official Oregon Software "Party Pascal" baseball hat, plus a bottle of Oregon wine with which the winners may begin a party at which they may wear their new hats. The winners are:

**Best Bug** Herje Wikegaard (SERN Dator Konsult)  
Lee Mattiesen (NW Oklahoma State U.)

**Application** Joann H. Schultz (Lockheed Electronics)

**Best Prize** Doug Foster (E-systems)



## This issue marks a transition

Oregon Software's seventh Pascal Newsletter marks a transition. David Spencer officially takes over as editor, and I step down. Actually, David has coordinated the writing and production of the last several issues. I have written a few articles and have assisted with the review and rewriting of others, but David has been doing the vast majority of the work.

David is also taking over as manager of the technical writing group. I am returning to the ranks of simply "writer." This changeover seems to be an appropriate time to review our company's writing efforts since July of 1980, when Oregon Software's first technical writer (yours truly) was hired.

We instituted the newsletter with the ultimate goal of quarterly issues. The one-man writing staff started slowly. After adding two staff members — David and Mike Kuhn, who has been our primary manual specialist — we have quickened the pace for a total of 7 in about 30 months. Time constraints still force us to combine issues on occasion, but the result has been the publication of newsletters with more substance. This one, for instance, is the combined Summer-Fall issue, with special emphasis on concurrent programming. It's the largest and most technically oriented issue this year.

We plan to continue with at least one staff-produced technical article and one user-produced article each issue. You should note that we run as many good user articles as we get, and we're always looking for more. Please write or call David with your ideas. We'll also continue with our regular fare of product announcements, OPUS columns, book reviews, letters, etc. We hope to have a more detailed Bug Log, including better lists of known bugs, both fixed and unfixed. Our biggest problem is that the production and mailing processes require several weeks' time, and the number of known bugs, and their status, can change dramatically after the newsletter has gone to press.

For our documentation in general, our goal has been to have manuals that reflect the high quality of our software. Further, we wanted a plain style that described complex software in the simplest terms possible. To a large degree, I believe we have succeeded. This summer marked the release of our new 2.1 manuals. Mike was the primary author of the new manuals, which not only document new software features but also provide much more detail on existing features. The considerable expansion of the 2.1 manuals represents about six months of Mike's hard work, with occasional help from the rest of the writers and a good deal of assistance (as always) from our programming staff.

In addition to manuals for new products, including SourceTools and several Pascal-2-based development systems, we are also working on new manuals for our original Pascal product, Pascal-1. A supplement to the existing Pascal-1 manuals is complete, and a revised manual similar to the Pascal-2 manuals should be done this fall. One of David's biggest tasks will be that of bringing all of our manuals up to the uniformly high standards of our Pascal-2 product line.

We need your help, by the way. Many of the improvements in our manuals have resulted from comments by users — a small but vocal group of users, as it happens. I'd like to take this last opportunity as outgoing editor to solicit comments and questions about our newsletter and documentation. Though our materials are reviewed thoroughly in-house before release, we can't really know whether they meet the needs of the users unless you tell us.

Collins Hemingway

---

## Pascal NEWSLETTER

---

OREGON SOFTWARE

2340 SW Canyon Road  
Portland, Oregon 97201

David Spencer, editor

Collins Hemingway and Michael Kuhn, staff writers

Erin Ryan, production assistant

The Pascal Newsletter is published quarterly by Oregon Software, Inc., 2340 SW Canyon Rd., Portland, OR 97201; (503) 226-7760. Each customer of Oregon Software receives one free subscription per site. Additional subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor. We pay \$100 per newsletter page for any article we print.

Copyright © 1983 by Oregon Software, Inc.  
ALL RIGHTS RESERVED.

RSTS, RSX, RT-11, PDP-11, VAX/VMS, and IAS are trademarks of Digital Equipment Corp. UNIX is a trademark of Western Electric. Pascal-1, Pascal-2, SourceTools and Pascal Newsletter are trademarks of Oregon Software.

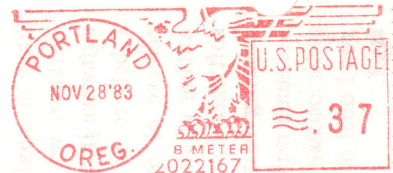
Printed in USA

---

# Pascal NEWSLETTER

OREGON SOFTWARE

2340 SW Canyon Road  
Portland, Oregon 97201



John Peterson  
Apt #714  
130 South, 13th East  
Salt Lake City, UT, 84102



# Pascal NEWSLETTER

NUMBER 9

OREGON SOFTWARE

FALL 1984

## Pascal-2 spans DEC line with VMS release

Oregon Software's highly optimizing Pascal-2 compiler is now available for the VMS operating system on the VAX and MicroVAX.

Release of the VMS native compiler makes Pascal-2 the only Pascal compiler that provides a uniform development environment across the full line of Digital systems from the Pro 350 to the VAX.

Unlike other language compilers available on DEC systems, all versions of Oregon Software's Pascal-2 provide identical language features, including the same interface with the operating system and hardware and the same set of language extensions. All versions contain identical programming development tools, including an interactive, high-level debugger.

Pascal-2 is integrated into the VAX/VMS system. The compiler allows calls to separately compiled routines written in Pascal, FORTRAN, or MACRO, giving developers access to existing software libraries.

In addition to DEC systems, Pascal-2 runs on M68000-based computers under the UNIX and VERSAdos operating systems.

For current Pascal-2 users, this means that applications code may be moved quickly and easily among many popular operating systems and that development work among these systems may be done with a uniform set of tools.

### Tools aid developers

Like Oregon Software's other compiler systems, Pascal-2 for VMS is designed to minimize the time users spend correcting errors and developing products.

Pascal-2/VMS includes the standard kit of software tools provided with all Pascal-2 systems. These include the interactive debugger, program and text formatters; cross-referencers for program identifiers and procedures; a dynamic string package written in standard Pascal; and a set of definitions to allow Pascal-like use of MACRO code.

Compile-time error checking ensures conformance of data types to the ISO standard, locates many uninitialized variables, and detects nearly 150 Pascal syntax errors. Comprehensive run-time checking detects array index errors, illegal subrange assignments, nil pointer references, non-existent case labels, and I/O and arithmetic errors.

In addition to generating error messages for run-time errors, the compiler produces a trace, or walkback, of the source statements leading to the error. Users are able to recover from normally fatal I/O errors and even to write their own I/O error-recovery code. Future releases of the Pascal-2/VMS compiler will provide the execution profiler and the capability to customize error messages for fatal run-time errors, which are already included in our other Pascal-2 systems. Run-time checking may be disabled for maximum performance.

### Pascal-2 supports standard, optimizes code

Pascal-2 for the VAX supports all features of standard Pascal, including packed data structures and set types of up to 256 elements. Pascal-2 supports integer types of 1 to 32 bits. Pascal-2 adheres to Level 1 of the international standard (ISO 7185), which includes conformant array parameters to provide dynamic arrays.

The Pascal-2 compiler performs eight types of code optimization designed to generate fast, small code: global register allocation, common subexpression elimination, expression targeting, array index simplification, range tracking, constant folding, dead code elimination, and short-circuit evaluation.

A compilation switch disables the compiler's extended language features, allowing only ISO standard Pascal programs to be compiled. The "standard" switch generates error messages giving the location of all non-standard code, simplifying the conversion of non-standard Pascal programs to Pascal-2.

---

## In this issue...

|                                         |         |
|-----------------------------------------|---------|
| Pascal-2 for VAX/VMS released . . . . . | Page 1  |
| Oregon Software changes . . . . .       | Page 2  |
| Information exchange . . . . .          | Page 3  |
| Who to call . . . . .                   | Page 3  |
| Sharing memory on RSX . . . . .         | Page 4  |
| More expansion . . . . .                | Page 7  |
| Errors, additions to manuals . . . . .  | Page 8  |
| The Log . . . . .                       | Page 11 |
| Known bugs . . . . .                    | Page 14 |
| Modula-2 Questionnaire . . . . .        | Page 16 |
| OPUS Directory . . . . .                | Page 17 |



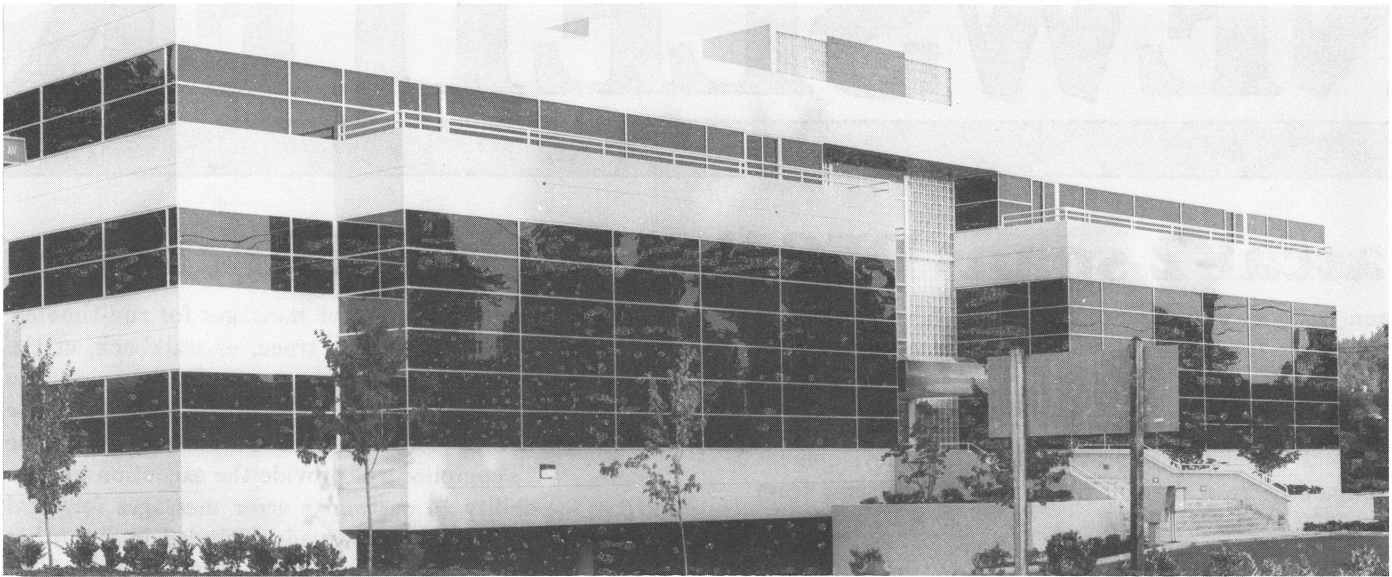


photo by David Spencer

**New location** – The entire second floor is devoted to staff offices, meeting rooms, a technical library, and the computer room. The employees' cafeteria, a large meeting room, and a large storage room occupy about a third of the bottom floor.

## Oregon Software begins new phase

A lot has happened at Oregon Software in the past few months. Perhaps the most exciting event for our staff has been our move to new offices on September 10th. We enjoyed our old building on Canyon Road for its casual atmosphere and for its location, near downtown and next to one of the largest city parks in the country. We did not enjoy the cramped conditions with 50 people crammed into our three separate buildings with a total of 9,000 square feet. Our new location is still near downtown, across the street from a very nice riverfront park, and the building gives us 15,500 square feet with plenty of room to grow.

We passed a major milestone in August when we placed Pascal-2 for VAX/VMS in field test. We are pleased to announce that this product will be available for delivery as of October 22, 1984. Most PDP-11 or M68000 Pascal-2 programs can now be recompiled to run in VAX native mode with little or no change.

We've heard from a number of our best customers that our product support has become inadequate. This news is very disturbing, and we are taking some definite steps to solve the problem. There are two areas of concern: bug-fixing and phone support.

We are working very hard to catch up on the backlog of trouble reports that has built up over the past year, especially for our cross-development software from the VAX to the Motorola M68000. We have hired some new programmers with compiler experience who are working hard to

learn Pascal-2 internals and to get problems fixed. The "old hands" on staff are giving them lots of help. For all products, we will provide maintenance releases twice a year, on a set schedule, as we do for our PDP-11 products.

We are also concentrating on teaching our phone support people more about our products and the programming environments in which they are used. Many of these staff members are relatively new to Oregon Software, and they are eager to learn more of the programming folklore that goes with our products.

We are confident that these steps will improve the quality of our technical support and we are grateful to our customers for their feedback on the level of service we provide.

*Don R. Baccus*

Don Baccus, President

## Come see us at Booth 2836

Oregon Software will be exhibiting products at DEXPO West, held at the Disneyland Hotel in Anaheim, California, December 11 - 14. Our sales staff will demonstrate new products, including Pascal-2 for VAX/VMS. Technical staff will answer your questions about existing products. We invite all our customers to visit us.



## Information exchange

If you need information on technical applications involving Pascal, or if you have an application that might interest other users, send us a brief description for inclusion in the Information Exchange. Your description should follow the format of the items below. Interested parties can contact one another directly.

**Two software tools** help programmers write correct, portable Pascal programs and help compiler writers produce accurate, bug-free, fully conformant standard Pascal compilers. The Pascal Validation Suite Quality Control Package (PVS) consists of 734 test programs which systematically exercise a Pascal compiler to determine its ability to process programs written in ISO standard Pascal. The Standard Pascal Model Implementation (SPMI) consists of a compiler and interpreter, used in combination to process the validation suite tests correctly, and a static checker which audits Pascal programs for conformity to the ISO standard. The entire SPMI implementation is written in Pascal. The PVS and SPMI are available in machine readable source code on tape and diskette. For more information, contact: Donna Kish, Software Consulting Services, Ben Franklin Technology Center 125, Murray H. Goodman Campus, Lehigh University, Bethlehem PA 18015, (215) 861-7920.

**PRM-11** is a Pascal Record Management system for use with Pascal-1 or Pascal-2 under RSX-11M V4.0. PRM-11 also runs under VAX/VMS V3.1 in compatibility mode, with the restriction that shared files may not be opened with **write** access. Documentation is included. Contact: Doug Bliss, Toledo Scale, 1150 Dearborn Drive, Columbus OH 43229, (614) 438-4877.

---

## OPUS Communiqué

---

*Oregon Pascal Users Society*

---

The column for this issue consists of the OPUS membership list for September 1984. We've arranged the pages at the back of the newsletter so that you can cut them along the inside edge, staple them in the middle, and make a 5.5 by 8 inch booklet. Further information will be in our winter issue.

## Who to call...

To be connected with the "right" person when you call Oregon Software, you can ask for one of the following people by name or tell the receptionist the type of information you want.

**Technical matters** – tell the receptionist that you need technical help. She will connect you with the support person for that day.

**Updates or support status** – ask for Customer Service Manager, Claire Lematta, or explain to the receptionist that you are calling about support renewal.

**Manual orders** – direct your request to Diane Likens.

**Purchases and product information** – ask for Sales or tell the receptionist what information you wish. Sales representatives Tim McMenamin, Penny Green, Dan Kane, and Lorie Griffith serve prospects and customers within the United States.

**International distributors** – For sales inquiries from outside the U.S., ask to speak to Linda Lausman.

**Domestic distributors and OEM accounts** – ask for Mary Erichsen, our Sales Manager.

---

## Pascal NEWSLETTER

---

OREGON SOFTWARE  
6915 SW Macadam Avenue  
Portland, Oregon 97219

**Editor** David Spencer

**Staff Writers**

Thomas E. Hanrahan, Mary Hutton, Louise Waitt

**Production Assistants**

Betsy Slonaker, Jennifer Mulder

The Pascal Newsletter is published quarterly by Oregon Software, Inc., 6915 SW Macadam Avenue, Portland, OR 97219; (503) 245-2202. Each customer of Oregon Software receives one free subscription per site. Additional subscriptions are available upon written request.

Copyright © 1984 by Oregon Software, Inc.  
ALL RIGHTS RESERVED Printed in USA

RSTS, RSX, RT-11, PDP-11, VAX/VMS, and IAS are trademarks of Digital Equipment Corp. UNIX is a trademark of Bell Laboratories. Pascal-1, Pascal-2, SourceTools, and Pascal Newsletter are trademarks of Oregon Software.

---

# Pascal tasks share memory under RSX

by Steve Poulsen

Even if your computer has a megabyte of memory, and even if your DEC sales representative says otherwise, no PDP-11 program accesses more than 64K bytes of memory at a time. The limitation is inherent in the PDP-11 architecture: Since addresses are 16 bits long, a single program can access only 65536 (64K) bytes or 32768 (32K) words of memory.

With large software projects, this may be a major inconvenience, because programmers must split up any large program into several smaller parts. The use of active page registers (APRs) to share memory between Pascal tasks on the RSX operating system allows these smaller parts to communicate.

First, some background. PDP-11 hardware registers are 16 bits wide, and instruction lengths are multiples of 16 bits. More important, memory addresses are stored as 16-bit values. On most PDP-11 computers, the memory management hardware is used to map virtual addresses to physical memory addresses. Virtual addresses, which are the 16-bit addresses used by a single program, are always in the range of 0 to 65535. For each virtual address, the memory management hardware computes an 18-bit or 22-bit physical address. Physical addresses may range from 0 to several million. The allocation of physical memory to tasks is a primary responsibility of the operating system.

You can use the APR in one task to map to the same physical addresses as an APR in another task, which allows the two tasks to share memory (data). This requires four steps, described in detail below: 1) create a special portion of physical memory called a partition to hold the data to be shared; 2) create a MACRO file that allows the Task Builder to map tasks to that partition; 3) install the task image for the partition; 4) use **loophole** to force Pascal variables to the start of the partition.

Before creating a partition, you must understand the allocation of virtual memory within a single task, which is controlled by 8 APRs. Each APR controls up to 8K bytes of memory. The following table shows the address ranges controlled by each APR.

The memory management hardware uses the upper three bits of the 16-bit virtual address to determine which APR (0 to 7) to use to map the physical address. The remaining 13 bits specify an offset in the range from 0 to 8191 bytes from the physical address mapped by the APR. Each APR has a length and an access mode associated with it. The length specifies the number of 64-byte blocks mapped by the register, and the access mode describes the kind of

references that can be made (read-write, read-only, or no access).

## Address Ranges Controlled by APRs

| Octal Addresses | APR |
|-----------------|-----|
| 000000-017777   | 0   |
| 020000-037777   | 1   |
| 040000-057777   | 2   |
| 060000-077777   | 3   |
| 100000-117777   | 4   |
| 120000-137777   | 5   |
| 140000-157777   | 6   |
| 160000-177777   | 7   |

Consider a program containing 30000 (octal) bytes or 6K words. For this task, the operating system sets APR 0 to map virtual addresses from 0 to 017777, its limit, and APR 1 is set to map addresses from 020000 to 027777, to make a total of 30000 bytes. Both APRs are set with read-write access automatically by the system. APRs 2 through 7 would be set to "no access." If the program attempts to access a virtual address in the range 040000 to 177777, a "memory protection violation" is generated. The allocation of virtual memory can be controlled by the Task Builder or by special system services called at run-time. A Pascal program allocates the heap and stack by asking the operating system to allocate additional virtual memory to the task as needed.

As an example, assume that you want to share 80 bytes of data between two Pascal programs. In particular, you want to write one program to leave a message in the shared memory area and a second one to retrieve and print the message.

You need to create a partition to hold the data. The partition specifies which area of physical memory is to be shared and prevents the operating system from using the memory for other purposes. Your system manager can create the partition dynamically or with VMR, so that the partition is installed each time the system is booted. In either case the commands are similar.

Your system is probably already making full use of available physical memory. To add a new partition, you need to reduce the size of an existing partition. This can be done with the SET/TOP command. The GEN partition is usually the largest partition, and it can usually be reduced in size without adverse side-effects. Use the PAR command to view the current partition allocation.

You want to create a common partition. Note that the GEN partition in Figure 1-A holds 515700 bytes and occupies physical memory in the range of 242100 to 760000 ( $760000 = 242100 + 515700$ ). You want to allocate 80 (decimal) bytes of data. Since partitions are always multiples of 64 bytes, round the number needed up to 128 bytes (or 200 bytes in octal). To make room for 200 bytes at the end of the GEN partition use the command:

```
>SET /TOP=GEN:-2
```

The -2 parameter means that the top of the GEN partition is reduced by 200 bytes. (Memory addresses and sizes are in multiples of 100 octal bytes, so the final two zeroes are dropped.) By reducing the size of the GEN partition, you leave room for your common area DATA in the range 757600 to 760000 ( $757600 = 760000 - 200$ ). To create the DATA common area, use the command:

```
>SET /MAIN=DATA:7576:2:COM
```

This creates a common partition called DATA based at 757600 with a length of 200 (octal) bytes. Use the PAR command to see the new partition (see Figure 1-B).

If you use VMR to create the DATA partition, reboot your system to make the partition available.

### MACRO file describes data

After creating the DATA partition, you must next create a MACRO file to specify the amount of data to be placed in the partition. This enables the Task Builder to map tasks to the partition. (The size of the data need not be the same size as the partition, though the partition must always be at least as large as the data. This is why both the partition and the MACRO program are needed. In the next example, the partition size is the same size as the data, but you could—wastefully—put a small amount of data in a large partition.)

To share the 80 bytes of data, create a MACRO source file called DATA.MAC that allocates 80 bytes of data. The .BLKB directive allocates 80 bytes of data.

```
.TITLE DATA
.BLKB 80.
```

```
.END
```

The decimal point after the 80 is required to interpret the number as a decimal number rather than an octal number. This file is assembled to produce DATA.OBJ.

```
>MAC DATA=DATA
```

```
>PAR
EXCOM1 067734 070000 014700 MAIN COM
EXCOM2 067670 104700 010200 MAIN COM
LDRPAR 067624 115100 002600 MAIN TASK
TTPAR 067260 117700 030000 MAIN TASK
DRVPAR 066734 147700 003700 MAIN SYS
        066670 147700 002300 SUB DRIVER -DL:
        066570 152200 001400 SUB DRIVER -DX:
SYSPAR 066470 153600 010100 MAIN TASK
FCSRES 066424 163700 032000 MAIN COM
FCPPAR 066360 215700 024200 MAIN SYS
        041204 215700 024200 SUB (F11ACP)
GEN 066314 242100 515700 MAIN SYS
        041600 242100 020000 SUB (...MCR)
```

A. Existing Partition

```
>PAR
EXCOM1 067734 070000 014700 MAIN COM
EXCOM2 067670 104700 010200 MAIN COM
LDRPAR 067624 115100 002600 MAIN TASK
TTPAR 067260 117700 030000 MAIN TASK
DRVPAR 066734 147700 003700 MAIN SYS
        066670 147700 002300 SUB DRIVER -DL:
        066570 152200 001400 SUB DRIVER -DX:
SYSPAR 066470 153600 010100 MAIN TASK
FCSRES 066424 163700 032000 MAIN COM
FCPPAR 066360 215700 024200 MAIN SYS
        041204 215700 024200 SUB (F11ACP)
GEN 066314 242100 515500 MAIN SYS
        041600 242100 020000 SUB (...MCR)
DATA 041430 757600 000200 MAIN COM
```

B. New Partition

## Figure 1. Partition Allocation

The first column is the name of the partition. The second column is the internal location of the partition control block within the monitor and is not too useful. The third column is the base address of each partition or subpartition. The fourth column is the length of the partition. The fifth column is the type of the partition. MAIN is the main partition while SUB is a subpartition of a MAIN partition. The sixth column shows that a partition can be a common area (COM), a system-controlled partition (SYS), a task-controlled partition, a device driver (DRIVER), or a device common (DEV).

### Installing the task image

The Task Builder creates the task image and symbol table for the partition. Several special options must be used. The `/-HD` option creates a task image without a task header, and the `STACK=0` option prevents the allocation of stack space in the task. The `PAR` option gives the name of the partition in which the task will be installed (DATA), the virtual base address of the partition (160000, the base address for APR 7) and the size of the partition (rounded up to 200 bytes). All numbers are expressed in octal. APR 7 is used to map the shared data because it is the highest available APR (see table). Since Pascal tasks may use additional APRs for the heap as they expand, it is best to use APRs at the high end of virtual memory. This provides the maximum memory area for the program.

You must use Task Builder commands to create three files: DATA.TSK, the task image for the partition; DATA.MAP, the map file; and DATA.SYM, the symbol table, which describes the partition and its contents. The symbol table is used by the Task Builder when Pascal programs are linked with the shared data area, as explained further below.

```
>TKB
TKB>DATA/-HD,DATA,DATA=DATA
TKB>/
ENTER OPTIONS:
TKB>STACK=0
TKB>PAR=DATA:160000:200
TKB>//
```

DATA.TSK should be installed with the INS command:

```
>INS DATA
```

### 'Loophole' forces data to virtual memory

You can now write Pascal programs that share the data. To specify which data is to be shared, you must force the Pascal variables to appear at virtual location 160000 (octal). Any data at that virtual location is mapped with APR 7. Then use a special Task Builder option to cause APR 7 to map to the DATA partition you created above.

Using Pascal-2, you can force data to be referenced at a particular virtual address by creating a pointer to the data. Then assign the pointer a virtual address with Pascal-2's `loophole` function.

Sample program WRITE shows how to force variables to appear at virtual location 16000 with the `loophole` function. The `type` statement does not allocate any memory for the `message` you want to share. It only describes what a message looks like. The `type message_pointer` is a pointer

to a message. The only data allocated to the program is `p` which is a pointer to a message. In the first statement of the program, the `loophole` function converts the octal integer 160000 to a `message_pointer` which can then be assigned to the pointer `p`. The second statement of the program prompts you to type a message and the third statement reads the message. Since the variable `p` is a pointer to a `message`, `p^` is the message itself. Note that the storage for the `message` is not contained in the program itself, but is allocated in the small MACRO program installed in the DATA partition.

```
program WriteMessage;
type
  message = packed array [1..80] of char;
  message_pointer = ^message;
var
  p: message_pointer;
begin
  p:=loophole(message_pointer,160000B);
  write('Type a message:');
  readln(p^);
end.
```

Compile the program WRITE.PAS:

```
>PAS WRITE
>TKB
TKB>WRITE/FP/CP,WRITE=WRITE,LB:[1,1]PASLIB/LB
TKB>/
ENTER OPTIONS:
TKB>RESCOM=DATA/RW
TKB>//
```

The RESCOM option specifies the location of the symbol table file, DATA.SYM, which describes the shared data area DATA. The `/RW` switch requests both read and write access to the common data area. The `/RO` switch can be used to request read-only access. If you copy DATA.TSK and DATA.SYM to LB:[1,1], then instead of RESCOM, use the COMMON option: `COMMON=DATA:RW`.

You can write a similar program to read the message stored in the common data area.

```
program ReadMessage;
type
  message = packed array [1..80] of char;
  message_pointer = ^message;
var
  p: message_pointer;
begin
  p:=loophole(message_pointer,160000B);
  writeln('Message: ',p^);
end.
```



Here, the pointer *p* is initialized as in the first program. The second statement prints out the message pointed to by *p*. Compile and build this program (READ.PAS) as follows:

```
>PAS READ
>TKB
TKB>READ/FP/CP, READ=READ, LB: [1,1] PASLIB/LB
TKB>/
ENTER OPTIONS:
TKB>RESCOM=DATA/RO
TKB>//
```

The RESCOM option causes the Task Builder to read the file DATA.SYM to obtain information about the shared common area called DATA.

Now you can use the WRITE program to write a message to the shared data area, and the READ program to read back the information.

```
>RUN WRITE
Type a message: this is a test
```

```
>RUN READ
Message: this is a test
```

```
>RUN WRITE
Type a message: so is this
```

```
>RUN READ
Message: so is this
```

```
>RUN READ
Message: so is this
```

Setting up a partition for the sharing of data and using *loophole* to reference it allows individual programs to access more than 32K words of *total* data, even though no more than 32K words of data may be accessed at any one instant. In practical terms, the maximum additional amount that can be shared is 24K words, because even the smallest Pascal program on RSX takes up about 8K words of memory.

## New staff adds to Oregon Software mix

Assistant Systems Manager, **Jim Anderson** joins Oregon Software after 17 years with Boeing Computer Services. In addition to his job, which Jim describes as "beautiful," he enjoys photography, playing guitar and banjo, writing poetry and working on hydroplanes and drag racers with his brother.

**David Barnes** joined the Technical Publications group this September. A graduate of Ohio State in communications theory, Dave has been a technical writer at a large company in Columbus for five years. He enjoys learning computer languages and has experience writing programs in Macro-10, BASIC, FORTRAN, PL/1, and BLISS. Dave claims "only a reading knowledge of Pascal and C" and looks forward to Pascal programming at Oregon Software. Living in the Pacific Northwest is a new experience for the Barnes family: "I'll have to learn to grow roses," Dave says.

Accounting Clerk, **Julie Berrigan** performs many tasks in our accounting department, especially processing and collecting accounts receivable. Her one-year-old, Matthew, keeps Julie running when she isn't at Oregon Software.

**JoAnn Bertram** became Department Secretary for Sales and Marketing in July. She claims to be the only University of Washington graduate who took ten years to get a BA in English literature; she was also putting her husband through school and taking care of two children at the time. Currently, JoAnn and her husband are remodeling their house.



photo by David Spencer

**Voices of Oregon Software** - Katie Wilding (left) has trained Rita Gorham (center) to take her place as receptionist. Sharon Lodewick (seated) serves as back-up on the phones.

**Denise Bristow** is improving customer support by organizing all trouble reports by product. She received a degree from Portland Community College in applications programming last March and joined Oregon Software in August. Denise has travelled extensively in Europe, North Africa and Hawaii. She likes jogging, rafting, hiking, mountain climbing and scuba diving. She hopes to take up racquetball again now that she's out of school.

Continued on Page 10

# Errors, additions to manuals

The Technical Writing staff keeps a list of changes and additions to be included in each manual. We receive information from readers through Documentation Evaluation Reports (the DER forms at the back of each manual), Trouble Reports, and support calls. Every six months, usually coinciding with a release of the software, we issue an update package.

The cycle of update packages is staggered. During the interim between updates, we list changes and additions in this newsletter and in the Release Notes.

## All Pascal-2 manuals

We've found the following errors in *Pascal-2 User Manuals*, Version 2.1 for RSX, RSTS/E, RT-11, and UNIX.

### A bug in a Debugger example

The description of the use of negative numbers as *count* parameters for the L command is incorrect. A negative *count* parameter causes the Debugger to list statements up to and including the statement number indicated in the command. The example should read:

```

} L(Rotate,4,-2)
  16      3      A[I] := A[I+1];
  17      4      A[Last] := A[First];

```

### Language Specification, extended-range arithmetic

Substitute the following passage for the existing third and fourth sentences of the paragraph:

Normal arithmetic operations, with the exception of division and modulo, are performed on extended-range variables. Comparisons and division are signed.

## Pascal-2 RSX manual

The *Pascal-2 User Manual, V2.1 for RSX*, contains the following errors.

### Overlays, page 2-19

In the TKB multiline command example, delete the third line; the /MP switch eliminates the need for this entry.

### Support library, page 2-21

Under "Support Library Data Definitions," the words "The file" (fourth line, first paragraph) should say "LIBDEF" to clarify which file defines the file name block.

### Multiple source files, page 2-53

In the source file EXAMPL.PAS the first `%include` direc-

tive (`%include 'config'`) should be deleted so the command at the bottom of the page does not include the file twice.

### Location of the compiler's work files, page 2-54

In the fourth paragraph, the "time" compilation switch should be "times," for completeness.

### Using event flags, page 2-67

The example program DELAY fails to take into account that an enumerated type with fewer than 256 members is allocated one byte. Potential problems arise when the enumerated parameter is passed to the `nonpascal` procedure MARK because MARK actually requires that a two-byte (word) parameter be passed. Defining the individual time intervals to be constant integers takes care of the problem because the size allocation for integers is two bytes. In the example, change:

```

type
  TimeInterval = (Ticks, Seconds,
                 Minutes, Hours)

```

to read:

```

const
  Ticks = 1;
  Seconds = 2;
  Minutes = 3;
  Hours = 4;

```

### Margins, page 5-48

In the third paragraph, the default value of the left margin (L10) should be L0. The value in the table immediately preceding that paragraph is correct.

### Pascal Procedures in Resident Library, pages 2-77,78

On each page, the number 5353 octal should be 5354.

### Existing Procedures in a Resident Library, page 2-79

The first full line of the task build command at the top of the page should read:

```

TKB>WRTSUM/FP/CP,WRTSUM=WRTSUM,LB:
      [1,1]PASLIB/LB:$FCSJT,LB:[1,1]PASLIB/LB

```

### 'Origin' Declaration, page 3-20

The reference to the section "Size Function" should be referencing the section "Size and Bitsize Function."

### Debugger Guide, pages 4-4 to 4-19

When beginning a debugging session, the 2.1B Debugger identifies the program being debugged in a slightly different

way than the 2.1A Debugger. Examples on these pages should show the corrected line: **Debugging program ROTAT.**

#### Example, page 5-34

In the 2\$: block of the MACRO-11 example, the variable **r0** in the comment at the end of the **bne** statement should be **n**.

In the last paragraph, the word **restore** should be **rsave**.

### Pascal-2 RSTS/E manual

The *Pascal-2 User Manual, V2.1 for RSTS/E*, contains the following errors.

#### Implementation of **escape** differs for RSTS

RSTS interprets the standard **escape** character **chr(27)** to be a line-terminator and returns a \$ prompt. To get a true escape character, you must set the high-order bit by using **chr(155)**.

#### Example: function 'NewOK', page 2-35

In the listing of **NewOK**, delete the reference to **\$\$HEAP** in the comment for the **space** function definition.

#### Error termination Status, page 2-50

In the second sentence of the note at the end of this section, the words "It is" should be inserted before the word "provided."

### Pascal-2 RT-11

The *Pascal-2 User Manual, Version 2.1 for RT-11*, contains the following errors.

#### Copyright, page ii

The trademark entries should include "TSX-Plus is a trademark of S &H Computer Systems, Inc."

#### Example: function 'NewOK', page 2-30

In the listing of **NewOK**, delete the reference to **\$\$HEAP** in the comment for the **space** function definition.

#### Multiple source files, page 2-44

In the fifth paragraph, first sentence, the second "are" should be deleted.

#### EXITST walkback for 'severe error' status (4)

The **Exitst** procedure is a support library routine that sets the termination status of a program and stops the program when a "severe error" status is detected. The procedure's integer argument determines the termination status for any program that calls it. When a "severe error" status

of 4 is passed, the procedure also invokes the post-mortem analyzer to create a walkback of the program execution from the point of failure.

### Pascal-2 UNIX manual

The *Pascal-2 User Manual, V2.1 for UNIX*, contains this error:

#### Storage allocation for packed record, page 2-17.

For 68000 users, the last line should read "beginning at bit 8 (most significant bit)."

### All Pascal-1 manuals

With the release of V1.3D, we'll issue a new user manual. In the meantime, the V1.2K manual and a supplement are the documentation.

#### Run-time checking switch is renamed

Oregon Software provides a set of notes entitled *Conversion From Pascal-1 to Pascal-2* for those customers upgrading from Pascal-1 V1.2 to Pascal-2 V2.1. If you have this document, please make this change. On page 1-7, you are told to change the embedded directive **\$A** to **\$arraycheck**. The correct change is to **\$indexcheck**.

#### Addition to supplement

Users should note that the *Supplement to Existing Manuals* for Version 1.3 does not describe a newly added feature: Pascal-1 Version 1.3 now accepts **\$** in identifiers.

### Pascal-2 VERSAdos V2.0 manual

The following items should be added to *Pascal-2 Version 2.0/M68000 User Manual*.

#### Debugger, page 126

The description of the use of negative numbers as *count* parameters for the **L** command is incorrect. A negative *count* parameter causes the Debugger to list statements up to and including the statement number indicated in the command. The example should read:

```

} L(Rotate,4,-2)
  16   3      A[I] := A[I+1];
  17   4      A[Last] := A[First];

```

#### Error termination status, page 37

The following text should be added to the section on "Run-Time Error Reporting."

Both the Pascal-2 compiler and Pascal programs return a termination status when they exit. The Pascal-2 compiler terminates with a "severe error" status if it detects

compilation errors. Upon detecting an error while running, such as "subscript out of bounds," a Pascal program also terminates with a "severe error" status. Otherwise, a "successful completion" status is returned.

The Pascal support library contains a routine that may be called from Pascal programs to set the termination status and stop the program. To use this feature, declare an external procedure named `exitst`. This procedure, defined in the support library, takes an integer argument, as shown:

```
procedure Exitst(Status: integer);
{ procedure declaration }
external;
```

Call the procedure at a point in the program where you want to exit in case of a severe error, as shown:

```
begin      { program Severe }
:
Exitst(4);  terminate with severe status
:
end.        { program Severe }
```

A status of 1 means normal termination; any other status means that an error terminated the program.

### Compilation Switch /longlib

In order to take advantage of "short references and instructions" which use 16-bit addresses, the Pascal-2 support library under VERSAdos resides in the low end (first 32K bytes) of memory. Support library subroutines use short references internally and the compiler generates calls to the support library using the 16-bit addresses.

The new /longlib switch causes the compiler to generate full 32-bit address calls to the support library. Users who wish to use /longlib must change all internal use of short instructions in the library's source code before relocating the support library.

## Concurrent Programming manual

Users of V1.0A should make this change.

### Booting Instructions, page 4-10

You are told to use the supplied program MAKEBOOT to convert the load module to a bootable image. Users must also compile and assemble FHSCAL.SA and FHSUTL.PA, two modules that are shipped with the utilities. These modules are then linked with MAKEBOOT.

## New staff

From Page 7

**Lyn Gabel**, Administrative Assistant to our President, has been writing a company personnel manual and will be doing research projects. A graduate of Oral Roberts University with a BA in psychology, Lyn says "I was a Marine brat, so we lived all over the U.S. when I was growing up." Lyn and her husband are Portland Winter Hawks boosters; they board one of the team's defensive players, a student from Canada, during the hockey season. Fortunately, Lyn "loves to cook."

**Rita Gorham** came from 4A's Temporary Service to fill in as our receptionist. She did such a good job that we asked her to stay. Rita enjoys cross-country bicycle touring and downhill skiing. She hopes to take up scuba diving so she can add seashells to her collection, preferably some from the Barrier Reef area of Australia.

Formerly an intern programmer, **Bruce Graham** became a full-time programmer in August. A graduate of Reed College in mathematics with additional coursework in computer science at OSU, Bruce has been working on the UNIX debugger. Before joining Oregon Software, Bruce spent five years in Alaska as a fire fighter and taught high school math and physics in New Zealand. He enjoys woodworking, particularly building boats.

Technical writer **Mary Hutton** joined Oregon Software in July and immediately began revising manuals for new releases of the software. A graduate of both Stanford (BA in Latin) and the University of Washington (BS in scientific and technical communication), her interests include archaeology, mathematics, language study, anthropology, and all types of crafts.

Sales representative **Dan Kane**, a Portland native, recently graduated from Oregon State University (BS degree in Business Administration) where he concentrated on marketing and sales management. Dan says he likes sales because it involves communicating with people. His avocations are playing as much golf as possible, playing softball, and skiing.

**Krzysztof Krüger**, a native of Poland, comes to Oregon Software via Aachen, Germany, where he programmed in Pascal-2 for three years. In addition to Polish and German, Kris is fluent in Russian, French, and English. After he gets his family settled, he hopes to resume his favorite hobbies, basketball and photography. What does he think of his new home? "I'm glad to be here," says Kris.

Continued on Page 15



# The Log: errors, work-arounds, changes

As in previous issues, this log also describes the significant changes in Pascal-1, Pascal-2, and SourceTools. As a service to users who may not have the current release, we supply work-arounds where possible.

For the first time, we've listed many known bugs: those problems reported by users and verified as bugs by Oregon Software. By conveying this information, we spare you the labor of tracing and reporting bugs unnecessarily and give you some indication of a particular report's status.

To use this log, you must know the release level of your software. (The release number is printed on the headline of all program listings.) Then review the log to determine the changes made since the release of your version. As an additional reference, we've placed the Trouble Report numbers in square brackets at the end of each log entry.

If the changes are of particular importance to your application, your Designated Contact Person should request an update, in writing. Upon receipt of your written request, you will receive the latest version of the software.

## Pascal-1

Version 1.3C has just been released. We have not received any Trouble Reports from users.

## Pascal-2

Version 2.1D corrects a number of problems common to PDP-11s with RSTS/E, RT-11, or RSX-11 operating systems.

### Packed boolean array problems

The compiler no longer generates incorrect code when a packed array of boolean crosses a byte boundary. Under V2.1C, only the lower byte of the word was fetched so bits representing the higher elements of the array were not set properly [1173, 1319, 1986].

Declared as a **var** parameter, the packed array of boolean is now passed to a procedure correctly [1269].

### Record field as a parameter

The compiler no longer generates bad code when certain nested record fields are passed as a parameter [1191].

### Negative stack offset for common subexpressions

Depending upon the location of a variable's declaration, two identical calculations involving certain functions, including sine and cosine, could have different results. The library routines for the functions no longer cause the com-

piler to reference a negative offset from the stack pointer [1572].

### Wrong results from integer comparison

A comparison between an integer and a subrange that is an element of a packed record no longer produces an incorrect result [1572].

### Incorrectly addressed real variable

Certain globally allocated real variables are now addressed correctly in large programs [1613].

### Dynamic string package errors

The **Insert** routine no longer prints the error message **Array subscript out of range** if the combined length of the two input strings exceeds the target string's length. Instead, the insert string is concatenated onto the end of the target string and truncated [1782].

The **Concatenate** routine no longer reports errors when the string to be concatenated overflows the target string. The input string is truncated to fit.

### %Page directive doesn't work

On listings, **%page** now increments the page number correctly [1914].

### Reserved instruction trap error

Using a **packed record** containing integers in a nested procedure call no longer causes a reserved instruction trap [1988].

### Utilities don't compile

With Versions 2.1A and 2.1B, attempts to compile the utilities resulted in an **Unknown Pascal run-time error** [2027, 2303a, 2393].

### Illegal instruction gives "compiler writer error"

Using certain illegal instructions resulted in the error message for internal problems instead of the appropriate error message [2042].

### Sets generate bad code

Set constructor expressions of the form **[var1..var2]** no longer generate bad code at times [2046, 2046a].

### No error message

The compiler now gives an error message for attempts to pass an element of a **packed** structure as a variable-parameter [2151].

### Incorrect use of 'R0' for procedure calls

The compiler now saves the contents of the R0 register before a call to a **nonpascal** procedure [2177a].

### Set type produces wrong results

The declaration and statement shown below now compile correctly [2177b].

```
type  a = set of (one,two,three);
var   b:a;

begin
    b:= [one,three];
    write(b * [] <= []);
end.
```

### Function returns incorrect result

The compiler now correctly changes bit lengths to byte lengths when a function returns a structured type [2177c].

### Bit optimizations incorrect

Compiler optimization of certain user-defined procedure calls no longer produces an incorrect sequence of execution [2184].

### PB formatter rejects 'nonpascal'

The formatter now recognizes the **nonpascal** directive and treats it exactly like the **external** directive [2202].

### String comparison range wrong

For string comparisons, **ord(char)** is now in the range 0..255 instead of -128..127 [2248].

### Unsigned characters treated as signed

The compiler no longer treats 8-bit characters as if they are signed numbers at times [2329].

### Assignment statements cause failure

A complicated series of assignment statements involving arrays of type **real** no longer fails. [2403].

### Compiler consistency checks reported

Version 2.1D fixed certain conditions causing the error message **Undeleted temps in procedure...** [1142].

### Compatibility mode address trap

The Task Builder does not give a start address to programs compiled without a main program and with the embedded **nomain** switch. When executed, such programs trap because no program may start at location 0 on the PDP-11. No error message is issued by the Task Builder or by the

Pascal-2 compiler, but the code generated is correct [1989].

## Changes to RT-11

### Misleading error message for nonexistent files

The compiler no longer prints the error message **Unknown Pascal run-time error...** for an attempt to compile a nonexistent file or a file containing a **%include** of a nonexistent file [2235].

## Changes to RSX

### /apd switch problems

Programs appending records to an existing file do not occasionally crash after crossing a block boundary. Formerly, these programs trapped out to RSX with **T-bit Trap** or **BPT executions** error message and changes in the size of the program caused a different error message [2392].

### 'Forini' incompatibility

The FORTRAN initialization routine, **forini**, now works properly on VAX in RSX compatibility mode [2370]. (This bug was not fixed in the V2.1C release, as reported in the last issue.)

### Memory protection violation

The value of R0 is now initialized before adding the offset [1273].

## Changes to UNIX

Version 2.1E is the current release of the UNIX/68000 compiler. We've corrected a number of problems in the V2.1D release and completed development work on the Debugger.

### Debugger

The Pascal-2 Debugger now functions under UNIX.

### Unnecessary checking in 'for' statement

The range-tracking optimization no longer performs unnecessary final-limit checks on generated code.

### Bad integer error message

Certain cases of incorrect code generated by **packed** fields in records have been corrected.

### Trap on exponent underflow

Support library routines now return a signed zero instead of causing a fatal run-time trap.

**Floating point division**

The entire divisor is now restored to the partial remainder, not just the low-order word of the divisor.

**Remainders and quotients inaccurate**

The correct register is now accessed for division of operands longer than 16 bits. Previously, quotients could be in error by one bit and remainders (**mod**) could suffer inexplicable loss of significance.

**Inaccurate results of 'If' conditional**

The expression '**if x mod y = 0**' no longer produces inaccurate responses.

**'For' loop failures**

For loops with a control variable that is an **enumerated** type of one byte in size now generate correct code [1918].

**Integer constant range checking**

Expressions of the form *integer-constant in [0..255]* no longer fail.

**Conformant array parameters**

Conformant array parameters in **forward** and **external** procedure declarations no longer take 25 minutes to compile [2311].

**Loss of stack pointer**

Very large programs no longer crash due to loss of the stack pointer.

**Real number accuracy**

Nested functions that return a **real** value no longer generate incorrect code occasionally.

**'Undeleted temps' error message**

Certain for loops no longer generate the error message **Undeleted temps in procedure ...** [2020].

**Order of declarations**

Declaring an **external** function before a global **var** declaration no longer generates an error message.

**Bad 'case' statement error message**

An untested value within a **case** statement now returns the error message **No case provided for this value** [2469].

**Changes to VERSAdos**

Version 2.0N is the current release of the VERSAdos na-

tive compiler. We've corrected some of problems that were in the initial release (V2.0M) and have V2.0P, containing more bug fixes, in field test. We've skipped an O-release because V2.0O is visually and verbally awkward.

**Debugger**

The Pascal-2 Debugger now runs under VERSAdos.

**External use of conformant array parameters**

The compiler no longer rejects external use of conformant array parameters.

**File access problem**

The compiler can now access files in directories other than the current one.

**Origin extension**

Variables used with the **origin** extension now generate correct code.

**Changes to cross-development systems**

The Cross-Development System's initial release for RSX-hosted systems was Version 2.0M in early 1983. Version 2.0M for VAX/VMS hosts was released later in the same year.

Version 2.0N corrected the following problems for cross-development systems hosted on both VMS and RSX.

**Conformant array index variable**

The compiler no longer generates bad code when passing a conformant array to a subroutine as a **var** parameter and attempting to access an element of the conformant array with an index variable that was not declared at the same nesting level as the conformant array parameter. Formerly, the wrong level of addressing was used during array index calculation; enabling checking had no effect on the error.

**External use of conformant array parameters**

The compiler no longer rejects external use of conformant array parameters.

**'For' loop with computed limits**

A for loop with computed limits now generates correct checking code when the index variable is within range.

**GLOBAL\$\$ constant**

For modules with a main program and a global data area greater than 32K bytes, the constant now generates correct code.

**/NOCHECK switch**

Checking can now be turned off; this switch also disables stack overflow checking.

**ORD operator**

When applied to a function of type **char**, the **ord** operator now generates correct code.

**Origin extension**

Variables used with the **origin** extension now generate correct code.

**Set types error message**

The compiler now generates the appropriate error message when encountering a set constant of base type **integer** which lies outside of the 0..255 subrange limit. Previously, it trapped out to the operating system with an access violation.

**Structured constant traps**

A duplicate definition of a structured constant produces an appropriate error message instead of causing the compiler to trap out to the operating system with an access violation.

**Changes to VMS-hosted system**

Version 2.0N, for VAX/VMS systems, corrected the following problems.

**Multidimensional conformant arrays**

The compiler no longer traps out to VMS with a divide by zero exception when compiling a two-dimensional conformant array parameter.

**Pack/Unpack**

The transfer features **pack** and **unpack** now function correctly when operating on **packed array of integer** subranges that are allocated 2 bytes per element: e.g., **packed array[1..5] of 0..1000**.

**SourceTools**

Version 1.0C is the current release. Newsletter #8 reported the bugs fixed in that release.

**Miscellaneous notes**

The Pascal Standard does not define the **mod** operation for negative divisors. If either operand of the function is negative, its result may be unexpected. Examples: **-30 mod**

**8** yields a result of 2, **-23 mod 8** yields 1, **-6 mod 5** yields 4. Programs must check for this condition.

**Known bugs**

We've not fixed all the trouble reports we've received for Pascal-2. Many bugs have been logged but not verified, which means we cannot categorize or describe them by type of problem. By the next newsletter, we intend to reduce the backlog to the point where this section is more informative.

In the meantime, we're listing those trouble reports that we've verified and scheduled for subsequent releases.

**Pascal-2 VERSAdos**

The following problems with Version 2.0N have been reported and verified for the native compiler.

**MOD function**

The **mod** function does not produce an error message for negative integers. (See the explanation under "Miscellaneous Notes")

**Statement numbers for input don't match output**

The statement number in the assembler output code (.SA extension) does not agree with program statement numbers when files are input with the **%include** switch.

**Utilities fail**

The utility programs may run out of memory on larger files. Relinking them may fix the problem.

**Pascal-2 cross-development systems**

The following problems with the V2.0N release have been reported and verified for VMS- and RSX-hosted systems.

**Bit test instruction gives illegal address**

The compiler generates an illegal addressing mode when using the bit test (BTST) instruction for a construct **a IN b** where **b** is a set expression evaluating to a constant at compile time and **a** is a variable.

**Cross-assembler faulty**

The OASYS Cross-Assembler occasionally rejects valid code.

**Error checking code**

The compiler does not generate checking code for subranges in the form **0..maxint**.



### Field width error with 'writeln'

At run-time, the expression `writeln(value:n)`; prints an indefinite number of blank lines under some conditions, e.g., printing hexadecimal output with the field specification (`value:-8`).

### For loop executes once only

A `for` statement in the form

`For integer:=integer div integer to integer div constant`

sets the loop index to the final value, so that the loop is executed once only.

### Size limit for arrays

The size limit set for certain declarations of arrays is 8M bytes instead of 16M bytes.

### Statement numbers for input don't match output

The statement number in the assembler output code (.SA extension) does not agree with program statement numbers when files are input with the `%include` switch.

### Truncated integers

No error message is generated if the value returned when an integer read from the standard input file (P\$4) must be truncated to fit in the storage allocated for the subrange.

### Unacceptable abbreviations

PASMAT accepts only the abbreviated form `/o` for the `/options` switch.

### Unacceptable file names

PASMAT does not accept long file-name arguments.

## Pascal-2 RSX-hosted system

The following problems with the V2.0N release have been reported and verified.

### Characters reversed

The characters of a string constant are reversed in a structured constant containing both strings and numbers.

### 'MOD' function

The `mod` function does not produce an error message for negative integers. (See the explanation under "Miscellaneous Notes")

## New staff

From Page 10

**Sharon Lodewick** spells Rita at our reception desk and is training to be a word processing operator. A native Oregonian, she graduated from Sunset High School last spring and will study American literature in the future, perhaps to become a writer or a teacher. She plans to go to Portland State part-time when she's not answering the Oregon Software phones.

**Lois McClain** became our Tele-marketing Representative in July after six years at American Data Services as a company liaison to financial institutions and two years as office manager for All West Display. She plans to learn more about programming at Oregon Software. An avid football fan, Lois "never misses a game" with her six-year-old son. Like Lynn, Lois comes from a military family and has lived in many western states. She collects Oriental porcelain figurines.

**Jennifer Mulder** worked during the summer as an Intern in the Technical Publications group. She has gone back to BYU for another year of college, including coursework in computers.

As secretary for the Sales Department, **Julie O'Brien** helps end users, distributors, our customer service and OEM sales staff, by sending out project correspondence and literature. Julie recently graduated from Portland State with a degree in social sciences and plans to do volunteer work at the Metro Crisis Center. A native of Oregon, Julie has three children and likes to play tennis.

**Cyndy Smith** has joined our Technical Department as secretary. A graduate of the University of Oregon in business management, Cyndy will be "taking care of the administrative details and handling special projects" for our programmers and software engineers. Cyndy just returned from three months touring Europe and two years living in Tsuruma, Japan, where her husband worked as an architect for a military contractor. Cyndy enjoys practicing the flower arranging she studied in Japan.

## Modula-2 ?

I'd like you to help us understand the growing interest in Modula-2 so that Oregon Software can better plan "if" and "how" we might approach delivering a Modula-2 product at some point in the future. This won't reduce our commitment to expanding our Pascal product lines and improving customer support.

Please telephone me at our free 800-874-8501 number or send me a photocopy of the following questionnaire with your responses plus any comments that you feel might help our planning. Ask anyone else in your organization with strong views on Modula-2 to respond as well.

1. What's your level of interest in Modula-2? Please write a short description or check the items that apply to you:

☐ a. I'm already using a Modula-2 compiler for \_\_\_\_\_

system. Provided by \_\_\_\_\_ Does it meet your needs? \_\_\_\_\_

☐ b. I plan to acquire a Modula-2 compiler. When? \_\_\_\_\_

☐ c. I'm interested in Modula-2 but have no specific plans.

☐ d. I'm not particularly interested in Modula-2.

2. What capabilities not in Pascal-2 would cause you to switch to Modula-2, C, Ada, or some other language?

3. If you could get a reliable, efficient Modula-2, would it be your first choice among languages? Rank the languages you would prefer to use.

4. What type of work might you do with Modula-2? Examples: process-control, real-time, teaching, systems programming, applications, utilities, business?

5. Rank the auxiliary packages you most want to see in any new language products? Examples: debuggers, profilers, construct editors.

6. What do you think about the view of some of our customers that a new language is not needed? They suggest that we further enhance our Concurrent Programming Package (CPP) to include more of the functionality of Modula-2.

Name \_\_\_\_\_ Title/Group \_\_\_\_\_

Company/Organization \_\_\_\_\_ License Number \_\_\_\_\_

Address \_\_\_\_\_

Telephone \_\_\_\_\_

Thanks in advance for helping with our product planning,



Rusty Whitney, Chairman

staple - - - - - staple

OPUS Directory  
1984

Mr. Matt Richards  
Research & Development  
Polkaudio  
1915 Annapolis Road  
Baltimore,  
MD 21230  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. A.B. Bailey  
Systems Consultant  
Centre-file Limited  
P.O. Box 177  
75 Leman Street  
London  
England E1 8EX  
GB 01-488 3131

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Manuel Guerra  
Experiencias Industriales,  
S.A.  
C/O Joaquin Rodrigo  
Aranjuez,  
Madrid  
SP (91) 891 0840

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RSX-11M  
APPLICATIONS: Business: Dept. Control  
SYSTEMS: OEM: Real-time control, ARQ  
Systems

Mr. Ed Moran  
Horace Mann School  
231 West 246th Street  
Bronx,  
NY 10471  
UE (212) 548-4000

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, 11/44  
OPERATING SYS.: RSTS/E  
APPLICATIONS: Academic  
SYSTEMS: Academic

Mr. Henry Baird  
RCA Lab  
Rm 201  
Princeton,  
NJ 08540  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Mike Dearing  
University of Wisconsin  
Inst. Sys. Center  
1500 Johnson Dr.  
Madison,  
WI 53706  
MW (608) 263-1564

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, VAX  
OPERATING SYS.: RT-11, RSX-11M  
APPLICATIONS: Scientific, Engrg.  
SYSTEMS: OEM, Academic, Research

Mr. Richard Chlopan  
Pros. Dir. Comp. Sci.  
Westmar College  
Le Mars,  
IA 51031  
MW (712) 546-7081 ext. 315

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, 11/44  
OPERATING SYS.: RSTS/E  
APPLICATIONS: Business: Admin. Programs,  
Academic: Comp. Sci. Proj.  
SYSTEMS: Academic: Monitors and Tools,  
etc.

Mr. B.J. Th. Ockeloen  
System Engineer  
Universiteit van Amsterdam  
Vakgroep Algemene Dierkunde  
(Dept. of Zoology)  
Biologisch Centrum Gebouw II  
Kruislaan 320  
1098 SM Amsterdam,  
Netherlands  
NE 020) 680551 Ext 133

COMPILERS: 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX, PDP 11/60, PDP  
11/34, MC68000 (soon)  
OPERATING SYS.: RSX-11M V.3.2 Autopatch E  
APPLICATIONS: Scientific: Zoological,  
Academic: Pattern recognition  
SYSTEMS: Academic

Mr. Sydney Davis  
Computrol  
15 Ethan Allen Highway  
Ridgefield,  
CONN 06877-6297  
UE (203) 544-9371

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Dr. A. Das  
Associate Professor  
Computer Science  
Quincy College  
Quincy,  
IL 62301  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Prof. Allan Smith  
Associate Professor  
Drexel University  
Department of Chemistry  
Philadelphia,  
PA 19104  
UE (215) 895-2667

COMPILERS: 1.2  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11  
APPLICATIONS: Scientific: Chem. Res.  
Academic: Instructional  
Software  
SYSTEMS:

Mr. Jeffrey Levine  
Department of Earth and  
Planetary Sciences  
The Johns Hopkins University  
Baltimore,  
MD 21218  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Robert A. Rennert  
Director of Academic Computing  
Kenyon College  
Gambier,  
OH 43022  
UE (614) 427-2244 Ext. 2559

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Murray Zetterholm  
Ford Motor Credit Co.  
The American Rd., Room 2235  
Dearborne,  
MI 48121  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Bob Friedman  
E.I. Dupont Company  
Clinical Systems Division  
Glasgow Site Rt 896  
Glasgow,  
DE 19702  
UE (302) 453-3280

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, PDP 11/70, 11/44,  
11/24, MC68000  
OPERATING SYS.: RSX-11M, RSX-11M/PLUS  
APPLICATIONS: General Sys. use Downloaders,  
Utilities, General Math  
SYSTEMS:

Mr. Robert Lacovara  
InterNet  
500 Grand Avenue  
Englewood,  
NJ 07631  
UE (201) 567-3363

COMPILERS: 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX /23  
OPERATING SYS.: RT-11, RSX-11M  
APPLICATIONS: Mfg. Data Keeping  
SYSTEMS: OEM, Bus.

Mr. Robert Thornton  
New York Institute of  
Technology  
Computer Graphics Laboratory  
P.O. Box 170  
Old Westbury,  
NY 11568  
UE (516) 686-7644

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:



Mr. Jim Nash  
JEOL USA Inc.  
11 Dearborne Rd.  
Peabody,  
MA 01960  
UE (617) 535-5900

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Alan Duberstein  
Pine Instrument Co.  
3345 Industrial Blvd.  
Bethel Park,  
PA 15102  
UE (412) 831-8870

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RSX-11M, RSX-11M/PLUS Versions  
APPLICATIONS:  
SYSTEMS:

Mr. Hal Laurent  
Psych Systems  
600 Reisterstown Rd.  
Baltimore,  
MD 21208  
UE (301) 486-2206

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Bob Schor  
The Rockefeller University  
1230 York Avenue  
New York,  
NY 10021-6399  
UE

COMPILERS: 1.1, 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11 (4 & 5), TSX-11 (2 & 3)  
APPLICATIONS: Scientific: Data Proc.,  
Academic: Algorithm Testing,  
Teaching  
SYSTEMS:

Mr. Steven Bentley  
Academic Computing Center  
Tougaloo College  
Tougaloo,  
MS 39174  
UM

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Ken Tibesar  
3M  
3M Center, Bldg. 18-1  
Saint Paul,  
MN 55144  
UM (612) 733-8819

COMPILERS: 1.1, 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX, VAX  
OPERATING SYS.: RSX-11M, RSX-11M/PLUS, VAX-VMS  
APPLICATIONS: Mfg.  
SYSTEMS: Process Control

Mr. Troy Monaghan  
William Rainey Harper College  
Roselle and Algonquin Roads  
Palatine,  
IL 60067  
UE (312) 397-3000

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Terry Medlin  
Vice President  
GEJAC Incorporated  
P.O. Box 188  
Riverdale,  
MD 20737  
UE (301) 8664-3700

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, VAX  
OPERATING SYS.: RSX-11M, VMS  
APPLICATIONS: Business  
SYSTEMS:

Dr. Richard J. Perry  
Villanova University  
Dept. of Electrical  
Engineering  
Villanova,  
PA 19085  
UE (215) 645-4969

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11 4.0  
APPLICATIONS: Academic: Communications,  
Control & Signal Proc.  
SYSTEMS: Academic

Mr. Harald Norvik  
IKU  
Organic Geochemistry Dept.  
Hakon Maynussons gt. 1B  
Postboks 1883  
Norway  
7001  
NO (7) 915660

COMPILERS: 1.2/1.3  
MACHINES: PDP 11/XX PDP 11/23, PDP-11/24  
OPERATING SYS.: RT-11 V.4 (slightly), RSX-11M  
V.3.2 & 4.0 (mainly)  
APPLICATIONS: Scientific: chemistry  
SYSTEMS: Business

Mr. Mike McCready  
Software Manager  
Automation Engineering  
Div. of Refrig. Eng. Co. Ltd.  
26 Great South Road, Otahuhu  
P.O. Box 12072  
Auckland,  
New Zealand  
NZ (09) 276-8524

COMPILERS: 1.2/1.3 Pascal-1  
MACHINES: PDP 11/XX /23  
OPERATING SYS.: RSM-11M  
APPLICATIONS: Real-time process control  
SYSTEMS: Real time process control

Ms. Tellern Asiado  
Institute of Advanced Computer  
Technology (I/ACT)  
SGV Development Center  
105 De la Rosa St.  
Legaspi Village,  
Makati  
Phillipines  
PH

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Herje Wikegard  
SERN  
DATOR KONSULT AB  
Box 14070  
Goteborg,  
Sweden  
SW +46-31-830800

COMPILERS: 1.3, 2.0/2.1  
MACHINES: PDP 11/XX, VAX, MC68000  
OPERATING SYS.: RT-11, RSX-11M  
APPLICATIONS: Scientific: Tech. Dev. Proj. as  
Consultants, Other: Real time  
sys.  
SYSTEMS: In house sys. or the customers  
own sys.

Mr. W.R. Mitchell  
Chief Civil Engineer  
THE HYDRO-ELECTRIC COMMISSION  
4-16 Elizabeth Street  
Hobart,  
Tasmania  
TA 30-1101

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Arthur Brown  
10709 Weymouth St.  
Garrett Park,  
MD 20896  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Paul Bauer  
Advanced Control Systems  
13809 Industrial Park Blvd.  
Minneapolis,  
MN 55441  
UE

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Bob Sanford  
Digital Equipment  
New Jersey Turnpike Authority  
c/o New Jersey Turnpike  
Authority  
Administration Bldg.  
P.O. Box 1121  
New Brunswick,  
NJ 08903  
UE (201) 247-0900 Ext 233

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX 11/60, 11/40 & 11/04  
OPERATING SYS.: RSX-11M  
APPLICATIONS: Automatic Traffic Surveillance  
& Control  
SYSTEMS: Real time

Mr. Gary Ware  
EECO Incorporated  
1601 East Chestnut Ave.  
Santa Ana,  
CA 92701  
UW (714) 835-6000

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Glenn Stearns  
Software Development Manager  
Microvertics Corp.  
1394 Shorebird Way  
Mountain View,  
CA 94043  
UW (415) 961-9454

COMPILERS: 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11 4.00B, RSX-11 4.0  
APPLICATIONS: Bus. Mgmt. Sys.  
SYSTEMS: Sell to end user

Mr. Chuck Campbell  
Diagnostic Engineering  
Digital Equipment Corp.  
301 Rockrimmon Blvd. South  
Colorado Springs,  
CO 80963  
UW

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. Peter Schmitz  
Software Engineer  
GenRad S.P.D.  
4620 N. 16th Street  
Phoenix,  
AZ 85016  
UW (602) 264-2475

COMPILERS: 1.2  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11  
APPLICATIONS: Scientific  
SYSTEMS: OEM

Rev. James Keene  
Dir. Computer Center  
Saint Louis University High  
School  
Backer Memorial  
4970 Oakland Avenue  
St. Louis,  
MO 63110  
UW

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX PDP 11/23  
OPERATING SYS.: RT-11, & TSX-PLUS  
APPLICATIONS: Academic: Teaching & In School  
Appl.  
SYSTEMS: Academic

Mr. Martin W. Sivula  
System's Manager  
Lunenburg Public Schools  
1079 Massachusetts Ave.  
Lunenburg,  
MA 01462  
UE (617) 582-9941 Ext 4

COMPILERS: 1.2/1.3  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11, RSTS/E  
APPLICATIONS: Scientific: Some, Business,  
Academic  
SYSTEMS: Academic

Mr. Phillip Ricker  
Engineering Test Manager  
Intel Corporation  
Components Division  
3585 S.W. 198th  
M/S F4-2-546  
Aloha,  
OR 97007  
UW (503) 642-6592

COMPILERS: 1.2/1.3  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11  
APPLICATIONS: Semi-conclusion Testing, Cont.  
Sys. Proc.  
SYSTEMS: In-House

Mr. Abe Getzler  
Systems Analyst  
Brooklyn Union Gas  
195 Montague St.  
Brooklyn,  
NY 11201  
UE (212) 403-2428

COMPILERS: 1.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RSX-11M, IAS  
APPLICATIONS: Mobile Dispatching  
SYSTEMS: Motorola

Mr. John Hallick  
Manager Partner  
Miller Lucas Company  
Systems & Programming  
4811 N. Sterling Ave.  
Peoria  
IL 61615  
UM (309) 692-5640

COMPILERS:  
MACHINES:  
OPERATING SYS.:  
APPLICATIONS:  
SYSTEMS:

Mr. John Norman  
Academic Computing  
Grinnell College  
Grinnell,  
IA 50112  
UM (515) 236-2570

COMPILERS: 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX 11/70  
OPERATING SYS.: RSTS/E  
APPLICATIONS: Coursework, Computer assisted  
instruction  
SYSTEMS: Academic

Mr. Pat D'Andrea  
Sr. Software Engineer  
MCC Powers  
2942 MacArthur Blvd.  
Northbrook,  
IL 60062  
UW (312) 272-9555

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX, MC68000  
OPERATING SYS.: RSX-11M  
APPLICATIONS: Energy Mgmt/ Temperature  
Controls  
SYSTEMS:

Mr. Jerry Pitzl  
Associate Professor  
MACALESTER COLLEGE  
1600 Grand Ave.  
Saint Paul,  
MN 55105  
UM (612) 696-6291

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/70, VAX (perhaps '83)  
OPERATING SYS.: RSTS/E  
APPLICATIONS: Academic  
SYSTEMS: Academic

Mr. Steven Brecher  
Software Supply  
4618 E. 6th Street  
Long Beach,  
CA 90814  
UW (213) 434-3723

COMPILERS: 1.2/1.3, 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RT-11 4.0, RSX-11M 4.0  
APPLICATIONS: Support of Oregon Software  
Customers  
SYSTEMS: OEM

Mr. Alex Neussendorfer  
Control Data Corp.  
3965 Meadowbrook Rd.  
St. Louis Park,  
MN 55426  
UM (612) 935-0366

COMPILERS: 2.1  
MACHINES: PDP 11/XX, PDP 11-34, PDP 11-44  
OPERATING SYS.: RSX-11M 4.0, 4.1  
APPLICATIONS: Business: (Printed Circuit  
Board Mfg.)  
SYSTEMS: Business

Mr. Ronald Webster  
Computer Services  
Arizona State University  
ECA 108  
Tempe,  
AZ 85287  
UW (602) 965-1203

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RSTS/E  
APPLICATIONS: Academic  
SYSTEMS: Academic

Mr. Scott Snadow  
General Dynamics  
P.O. Box 2507  
Mail Zone 4-68  
Pomona,  
CA 91769  
UM (714) 620-7511 Ext 4779

COMPILERS: 2.0/2.1  
MACHINES: PDP 11/XX  
OPERATING SYS.: RSX-11M  
APPLICATIONS: Scientific: Various, Other:  
Sys.-Related  
SYSTEMS:

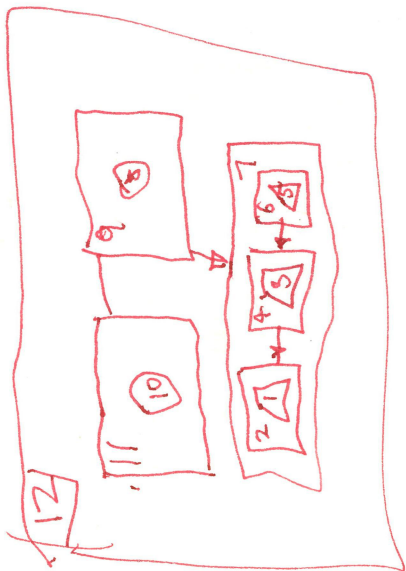
Ms. Cynthia Dunn  
Applied Automation, Inc.  
Pawhuska Road 205 ARB  
Bartlesville,  
OK 74004  
UM (918) 661-1915

COMPILERS: 2.0  
MACHINES: MC68000-EXORmacs  
OPERATING SYS.: VERSAdos  
APPLICATIONS: Scientific  
SYSTEMS: EOM

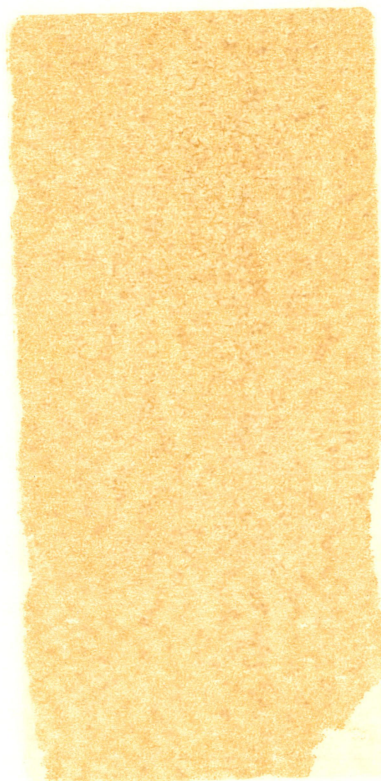


Phone Support  
 Library of Answers  
 Library of Answers

list  
 to use w/ wc  
 waste file  
 Find Bug, C# code  
 comp waste  
 CLEAN OFF EX WASTE



~~12~~





# Pascal NEWSLETTER

NUMBER 10

OREGON SOFTWARE

SPRING/SUMMER 1985

## ULTRIX compiler joins the Pascal—2 family

The last six months have been an intense period of product development at Oregon Software.

Recent developments include the completion of Pascal—2 for VAX/ULTRIX-32; completion of the Oregon Linker/Assembler for our cross-development systems to the 68000; and completion of native SourceTools for VMS.

Once Pascal—2 for VMS was completed in the fall, we were able to quickly produce the ULTRIX compiler. Our technical team, led by Bruce Graham, used the compiler's common front end with the VAX/VMS code generator to produce a compiler that was retargeted for the ULTRIX system. With this cross-compiler, Bruce then compiled the M68000 UNIX support library, which is itself written in Pascal. This produced a Pascal support library for ULTRIX, which provided a testing environment for larger, more complicated programs. The ULTRIX product was put into field test in April and is being released to end users now.

Completion of the Oregon Linker/Assembler frees us from reliance upon the Oasys cross-linker/assembler,

### VAX/VMS compiler retargeted for ULTRIX

which because of its slowness and incompatibility with standard M68000 object format was a weak link in our cross-tools packages from VMS and RSX. The Oregon Linker/Assembler runs under RSX and VMS native mode, making the cross-tools package available on all RSX and VMS configurations. In addition, our Linker/Assembler is designed to be largely machine-independent so that we can move it to new configurations without having to rely on other vendors' software in the future. In addition, the new versions of the cross-compiler itself (2.0P, shortly to be followed by 2.0Q) contain a number of bug fixes.

Native VMS SourceTools also completed field-testing and is being moved into regular distribution. Customers who purchased the RSX-to-VMS upgrade should be receiving their VMS versions shortly, if they haven't already.

We also completed the last round of bug fixes on Pascal—1, which is now an unsupported product. (See "Pascal—1 goes unsupported" later in this newsletter for update details.)

### Long-term development

A major goal of Oregon Software has been to make our products — as well as our customers' — portable. Because most of our products are written in Pascal, we have largely succeeded. Our efforts to quickly move Pascal—2 to a variety of different machine architectures, however, have sometimes resulted in compromise. Over time, compiler sources have drifted apart as unique machine-oriented features and bug fixes have been added. For example, in order to make our compilers interface in a natural way with various operating systems, we have had to modify or rewrite command-line processing code for each machine. This has gradually increased maintenance and development problems.

Over the last few months, our compiler developers have been working on the "Common Front End" project. Their goal is to realign the source code for the compiler to provide a clear distinction

between machine-dependent and machine-independent code.

The "front end" of the compiler is the code that scans Pascal source programs, analyzes syntax, and does the machine-independent optimizations. The "back end" of the compiler is the code generator, which is unique to each machine. The code generators for our different products are now being interfaced with the new common front end.

The common front end code served as the basis for the VAX/ULTRIX and the new 80186/286 project. Now that our compilers have converged on a common set of sources, maintenance and development will be much easier. Any new feature or bug fix in the front end code will immediately become available in each of the other compilers.

We began the 80186/286 project in April and now have a Pascal—2 compiler

### 'Common front end' improves maintenance and development

for the IBM PC/AT running XENIX. We have a large memory model for code size that takes advantage of the expanded instruction set of the 80286 processor and we are currently working on the code generation for 32-bit integer arithmetic. Our goal is to have an AT-based compiler

*Continued on Page 14*

## In this issue . . .

|                                                            |         |
|------------------------------------------------------------|---------|
| President's message . . . . .                              | Page 2  |
| Pascal—1 goes unsupported . . . . .                        | Page 3  |
| Using virtual memory with Pascal programs . . . . .        | Page 4  |
| Oregon Software documentation . . . . .                    | Page 8  |
| Book Review . . . . .                                      | Page 9  |
| Calling VMS system and run-time library routines . . . . . | Page 10 |
| Errors, additions to the manuals . . . . .                 | Page 12 |
| The Log . . . . .                                          | Page 15 |
| Newsletter Index . . . . .                                 | Page 21 |

# Common front end fosters modularity

Oregon Software's technical staff has been hard at work the past few months. Perhaps most exciting has been the work done to consolidate the various Pascal—2 compilers into one uniform structure, isolating machine-dependent code and sharing source modules to a much larger extent than before.

This work is significant for two reasons. First, expanding the use of common source modules will speed bug fixing, increasing the reliability of our Pascal—2 products. A good example is our recent effort to make Pascal—2 comply 100 percent with the ISO standard (past versions have been very close, but not perfect). As a result of the shared source code, most of our Pascal—2 products can be made to comply at a fraction of the effort necessary in the past.

This work will also lessen the effort required to introduce new compiler pro-

ducts. For example, Oregon Software recently announced availability of Pascal—2 for VAX/ULTRIX. The VAX/VMS compiler and VAX/ULTRIX compiler differ in only two areas: the user interface (program/debug vs. program -debug), and the object/assembly code output routines for the code generator. We have truly modularized the compiler and have made heavy use of MAKE (a component of SourceTools) to manage the system. For instance, the VAX/ULTRIX project began with the implementation of a VAX/VMS to VAX/ULTRIX cross-compiler. After this compiler was completed, it only required slight modification of the MAKE file to replace VMS-host specific modules with ULTRIX-host specific modules to generate a VAX/ULTRIX-hosted native compiler.

Likewise, our Intel 80186/286 pro-

duct (which has just entered field test) currently runs in native mode under PC-IX, as a cross-compiler under VAX/VMS, and as a cross-compiler under VAX/ULTRIX. These three versions are built by simply including the right pieces for each environment.

The net result of this effort? Better maintainability, as well as a quicker pace in development of new host-target environments supported by Pascal—2, which together mean better service to our customers.

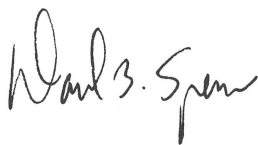


Don R. Baccus, President

## New editor named

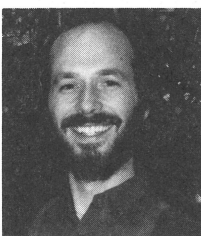
We're changing editors again! Collins Hemingway began the *Oregon Software Pascal Newsletter*. In the fall of 1983, I took over from him (Number 7). With this issue, Tom Hanrahan becomes our newsletter's editor.

Tom has worked on our user manuals since February 1984 and, until recently, he was managing editor of the *Portland Family Calendar*, a very popular monthly newspaper. Tom has produced this issue in a new format and included an index to previous issues. He has some good ideas for forthcoming issues, too. But, one thing never changes: he'll need articles about your applications, your technical tricks and discoveries, your Pascal-related interests. We can't have a users' newsletter without submissions from users.



David B. Spencer

## New employees draw on wide experience



Software engineer **Jan de Rie** joined Oregon Software in December. He comes to us from the Netherlands, where he worked for almost four years in Leiden.

There, he co-introduced Pascal—2 in a large organization (more than 100 programmers) and implemented the Pascal—2 support library on the company's proprietary operating system. Jan attended Erasmus University in Rotterdam, where he earned an M.A. in mathematical economics. Jan's main interest at Oregon Software is working with the Pascal compilers. Currently, he is developing a version of the Pascal—2 compiler for the Intel 80186/286 family of processors. Jan actively pursues his hobby of folkdancing and performs with a dance group around the northwest area.

As our newest account executive, **Ruby Houghton** will soon be introducing herself to many of our customers. Ruby came to Oregon Software in May from the sales department at Proactive, a division of Pacific Telecom Co. A graduate from the University of Oregon, Ruby brings with her the skills and experience

needed to provide new prospects with technical product descriptions and to respond to our current customers' requests for help and additional product information. Outside of Oregon Software, Ruby works with adolescents as a volunteer counselor, and she enjoys singing. She'll be touring the Caribbean this summer as a counselor/performer of an international sports evangelical team.

**Steve Bihler** joined Oregon Software in mid-January as Technical Support Manager. His primary function will be assisting customers with any technical problems they may encounter.

Steve's previous job was as a Senior Systems Engineer at Perkin-Elmer's Technical Support Center. He has an interdisciplinary degree from Washington State University in computer science and business administration. He likes flying, skiing, and radio communications. Steve has lived in more than 24 different places around the world including Greece and the Azores, and he has traveled extensively through Europe and the Orient.





## OPUS Communiqué

### *Oregon Pascal Users Society*

Bruce Williams, who served as the OPUS coordinator since the group's formation in November 1982, has given up his position as head of the Society. Any OPUS member interested in taking over as the Society's coordinator should contact:

Tim McMenamin  
Oregon Software  
6915 SW Macadam Ave.  
Portland, OR 97219  
(503)245-2202

The principal duties of the OPUS coordinator are to:

- Collect information submitted by OPUS members and disseminate it through this column.
- Facilitate contact among OPUS members by matching one member's problems with another member's expertise.
- Maintain and coordinate the OPUS shared library.

Over the past few years, OPUS has accomplished many of its initial goals. Membership has grown to over 100 individuals from around the world. The Society has helped numerous users of Pascal—1 and Pascal—2 with programming problems—everything from simple character I/O to asynchronous-trap routines, and most recently, the Society has taken steps to establish a shared library of routines and utility programs.

But clearly, the continued success of the Oregon Pascal User Society depends on the efforts of its membership and its ability to draw upon its membership to fill key positions within the organization.

## Pascal—1 goes unsupported

Pascal—1 version 1.3D was released for field test on April 8, 1985. Version 1.3D is the final release of our Pascal—1 compiler. The software is stable in its present form, and Oregon Software plans no subsequent releases.

Pascal—1 is Oregon Software's first Pascal compiler. Written entirely in MACRO-11, the compiler runs on DEC's PDP-11 series machines. It implements Pascal in much the same form as originally conceived by Jensen and Wirth with some extensions, such as separate compilation facilities.

Pascal—1 has been in commercial use since 1977. Thanks to its quick compilation and its minimal memory demands, Pascal—1 was rapidly embraced by the academic community, where it still enjoys wide use.

In the interest of portability,

improved code optimization, and adherence to the international Pascal standard (ISO 7185), Oregon Software introduced its new compiler, Pascal—2, in summer 1981. As Pascal—2 far exceeds Pascal—1 in performance and portability, we propose to devote our resources to maintaining and enhancing the newer compiler.

Oregon Software will no longer sell support contracts for Pascal—1. Customers who were under support for Pascal—1 as of January 1984 are eligible to receive the final update free of charge and should contact our customer service department to request their upgrade. Customers who are not eligible for the free update and wish to purchase the software should call our sales staff for price information and product availability.

## Pascal NEWSLETTER

OREGON SOFTWARE

6915 SW Macadam Avenue  
Portland, Oregon 97219

Thomas E. Hanrahan, editor  
David Spencer, David Barnes, Collins Hemingway, writers  
Betsy Slonaker, production assistant

The Pascal Newsletter is published regularly by Oregon Software, Inc., 6915 SW Macadam Ave., Portland, OR 97219; (503) 245-2202. Each customer of Oregon Software receives one free subscription per site. Additional subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor. We pay \$100 per newsletter page for any article we print.

Copyright © 1985 by Oregon Software, Inc.  
ALL RIGHTS RESERVED

RSX, RSTS, RT-11, PDP-11, VMS, ULTRIX, and VAX are trademarks of Digital Equipment Corp. UNIX is a trademark of AT&T Bell Laboratories, Incorporated. Pascal—1, Pascal—2, SourceTools and Pascal Newsletter are trademarks of Oregon Software.

Printed in USA

# RSX virtual memory window speeds record access

by Steve Poulsen

RSX system services known as memory management directives control the way in which memory management hardware of a PDP-11 computer maps a program's virtual memory address space to physical memory. In general, Pascal programs running on a mapped RSX system have access to a maximum of 64K bytes of memory. The limitation is imposed by the 16-bit nature of PDP-11 computers — at any instant, a program may not have direct access to more than 64K bytes of virtual address space. But physical memory on most PDP-11 computers is considerably larger than 64K bytes and as long as you do not attempt to directly reference more than 64K bytes at a time, you can write Pascal programs that call upon the memory management directives to take advantage of additional memory resources.

The physical memory accessible to a task is called a region. (Shared libraries and shared common areas are examples of regions.) The block of physical memory in which a task runs is called the "task region." Access to regions is through "windows," which are a portion of a task's virtual memory. Each window must start at an address that is a multiple of 4K words. So, in any 32K word task,

## Programs can access additional memory on most PDP-11s

up to 8 windows may be defined. The size of a window may vary from 32 words to 32K words. If a region is larger than the size of a window, the window may be moved to different locations within the region. For example, a 4K word window may be mapped to many different physical addresses within a 100K word region. The physical size of a region is limited only by the size of the partition in which the region resides.

To use virtual memory, a Pascal program must call upon the RSX memory management directives to:

- Attach to an exiting or dynamically created region.

- Create an address window to use in accessing a region.
- Map a window into a region.

The RSX memory management directives (described in the *RSX-11M/PLUS Executive Reference Manual*) may be called directly by a Pascal program through FORTRAN entry points in your system library (LB:[1,1]SYSLIB.OLB). To provide an interface between a Pascal program and the RSX directives, you must create two special data structures: a region definition block and a window definition block, both of which may be represented as simple Pascal records. The fields for these two structures correspond to the block parameters defined for specific memory management directives described in the executive reference manual.

The sample program VIRT.PAS, presented on the following page, is an example of how to call RSX memory management directives from a Pascal program. Explanations within the text of the code appear as ordinary Pascal comments. As the example shows, the program contains Pascal record definitions for the region definition and window definition blocks used by the system services. In addition, VIRT.PAS contains the nonpascal procedure declarations required in calls to the memory management directives.

The subroutines in VIRT.PAS implement a simple virtual file system by creating a dynamic region and then reading and writing data records within the region in much the same way that a disk file operates. The VINIT procedure initializes the virtual memory system by creating a dynamic region called PASDAT in the GEN partition. The user specifies the size of the region. VINIT also creates an address window that maps APR7 into the dynamic region. This causes virtual addresses in the range 160000 to 177777 to be mapped into the dynamic region.

To use VIRT.PAS, you must first modify the code to define the type of data to be stored in the virtual file. An integer is used for demonstration purposes in the current example. Access to the records in virtual memory is through the procedures VSEEK, VGET, and

VPUT. These procedures each use a record number to compute the physical location of the desired record in the dynamic region. Then, if necessary, a window is mapped into the region so that the desired record is contained within the window. This makes the record available to the task. The data can be read, updated in place, or copied to some other record in the task. In its current form, the VIRT program must be compiled

## Virtual file systems are very efficient

separately from its calling program because it uses OWN storage to maintain the current mapping information. The routine can easily be adapted for use as an include module by removing the \$OWN directive.

The Task Builder cannot predict how many windows your program might create. So, when you task-build a program that uses virtual memory, you must instruct the Task Builder to build additional window data structures. The WNDWS = *n* option creates additional window blocks in the task's header. For example, if your program is called TEST, you could task-build it with the following commands:

```
> TKB
TKB > TEST/FP/CP = TEST,VIRT,
TKB > LB:[1,1]PASLIB/LB
TKB > /
ENTER OPTIONS:
TK > WNDWS = 1
TKB > //
```

When you create windows, you should base them at high virtual addresses. In the current example, APR 7 (the highest available APR) is used for the mapping. If your task maps to a resident library such as PASRES, you may need to use a tower APR for the window because Pascal tasks extend the size of the task (window 0) when additional heap space is required. When you are also using virtual memory for data or for virtual overlays, it is possible that the program may accidentally extend into virtual address already being used for other purposes.



```
PROGRAM virt;
```

```
CONST
```

```
  apr = 7; { APR to use for mapping }
  blockettes_per_4kw = 128; { 8192 div 64 }
```

```
TYPE
```

```
  data = integer; { Simple test case }
  data_pointer = ^data; { Pointer to data record }
  byte = 0..255; { Unsigned byte }
  word = 0..65535; { Unsigned word }
  name = PACKED ARRAY [1..6] OF char; { Name to convert to rad50 }
  rad50_name = ARRAY [1..2] OF word; { 6 rad50 chars }
  directive_status = word; { Directive status returned from exec call }
  region_id = word; { System supplied region ID }
  region_status_bits = (rs$red, rs$wrt, rs$ext, rs$del, rs$hex, rs$att,
    rs$ndi, rs$mdl, rs$xx1, rs$xx2, rs$xx3, rs$xx4,
    rs$xx5, rs$xx6, rs$unm, rs$crn);
  region_status = PACKED SET OF region_status_bits;
  access_bits = (pread, pwrite, extend, delete);
  access_category = (system, owner, group, world);
  accesses = PACKED SET OF access_bits;
  protection_word = PACKED ARRAY [access_category] OF accesses;
```

```
  region_block =
```

```
    RECORD
```

```
      gid: region_id; { System supplied region id }
      gsiz: word; { Size of region in blockettes [64 bytes each] }
      gnam: rad50_name; { Region name [or zeroes] }
      gpar: rad50_name; { Partition in which region resides }
      gsts: region_status; { Region control & status bits }
      gpro: protection_word; { Region protection word }
    END;
```

```
  window_status_bits = (ws$red, ws$wrt, ws$ext, ws$del, ws$bpts, ws$sis,
    ws$rcx, ws$map, ws$64b, ws$nat, ws$res, ws$bnp,
    ws$rrf, ws$elw, ws$unm, ws$crw);
  window_status = PACKED SET OF window_status_bits;
```

```
  window_block =
  PACKED RECORD
```

```
    nid: byte; { System supplied window id }
    napr: byte; { APR to use to map virtual addresses }
    nbas: word; { Virtual base address of window [APR*8192] }
    nsiz: word; { Size of window }
    nrld: region_id; { ID of region which this window maps into }
    noff: word; { Offset [in blockettes] into region }
    nlen: word; { Length [in blockettes] to map [0 = default] }
    nsts: window_status; { Window control & status bits }
    nsrb: word; { Send/receive buffer address [not used] }
  END;
```

```
VAR
```

```
  data_size: word; { Size in bytes of DATA }
  max_record: word; { Max record number }
  records_per_4kw: word; { Number of data records in each 4KW block }
  current_4kw_block: word; { Block currently mapped }
  rdb: region_block; { Region definition block }
  wdb: window_block; { Window definition block }
```

```
PROCEDURE atrg(VAR rdb: region_block;
  VAR status: directive_status);
  NONPASCAL; { Attach region }
```

```
PROCEDURE crng(VAR rdb: region_block;
  VAR status: directive_status);
  NONPASCAL; { Create region }
```

```
PROCEDURE dtng(VAR rdb: region_block;
  VAR status: directive_status);
  NONPASCAL; { Detach region }
```

```
PROCEDURE caw(VAR wdb: window_block;
  VAR status: directive_status);
  NONPASCAL; { Create address window }
```

```
PROCEDURE elaw(VAR wdb: window_block;
  VAR status: directive_status);
  NONPASCAL; { Eliminate address window }
```

```
PROCEDURE map(VAR wdb: window_block;
  VAR status: directive_status);
  NONPASCAL; { Map address window }
```

```
PROCEDURE umap(VAR wdb: window_block;
  VAR status: directive_status);
  NONPASCAL; { Unmap address window }
```

```
PROCEDURE cvt_rad50(n: name;
  VAR r: rad50_name);
```

```
FUNCTION rad50(a, b, c: char): word;
```

```
FUNCTION rad(c: char): word;
```

```
BEGIN { Rad }
```

```
  IF c = '' THEN rad := 0
  ELSE IF c IN ['A'..'Z'] THEN rad := ord(c) - ord('A') + 1
  ELSE IF c IN ['0'..'9'] THEN rad := ord(c) - ord('0') + 30
  ELSE IF c = '$' THEN rad := 27
  ELSE IF c = '.' THEN rad := 28
  ELSE IF c = '_' THEN rad := 29
  ELSE
```

```
    BEGIN
```

```
      writeln('*', c, '* is not a legal RAD50 character!');
```

```
      rad := 0;
```

```
    END;
```

```
END; { Rad }
```

```
BEGIN { Rad50 }
```

```
  rad50 := (rad(a) * 40 + rad(b)) * 40 + rad(c);
```

```
END; { Rad50 }
```

```
BEGIN { Cvt_Rad50 }
```

```
  r[1] := rad50(r[1], r[2], r[3]);
```

```
  r[2] := rad50(r[4], r[5], r[6]);
```

```
END; { Cvt_Rad50 }
```

*The program VIRT.PAS provides an example of how to use virtual memory from Pascal. The subroutines in VIRT.PAS implement a virtual file system that is more efficient and has faster accessibility than a similar file system maintained on disk.*

```

PROCEDURE vinit(number_of_records: word);
EXTERNAL;

PROCEDURE vinit;

VAR
  n_4kw_blocks: word; { Number of 4KW blocks needed to hold virtual data }
  remainder: word; { Blockettes required in final portion }
  status: directive_status; { Directive status }

BEGIN { Vinit }
  data_size := size[data];
  IF odd(data_size) THEN data_size := data_size + 1;
  max_record := number_of_records;
  records_per_4kw := 8192 DIV data_size;
  n_4kw_blocks := number_of_records DIV records_per_4kw;
  remainder := ((number_of_records MOD records_per_4kw) * data_size +
    63) DIV 64;

  WITH rdb DO
    BEGIN
      gid := 0;
      gsiz := n_4kw_blocks * blockettes_per_4kw + remainder;
      cvt_rad50('PASDAT', gnam); { Region name }
      cvt_rad50('GEN', gpar); { Partition containing region }
      gsts := [rs$att, rs$wrt, rs$red, rs$mdl];
      gpro[system] := [ ];
      gpro[owner] := [ ];
      gpro[group] := [delete, extend, pwrite];
      gpro[word] := [delete, extend, pwrite, pread];
    END;
  crng(rdb, status);
  IF status < > 1 THEN
    writeln('Can''t create region. Status = ', status: 1, ' . ');
  WITH wdb DO
    BEGIN
      nid := 0;
      napr := apr;
      nbas := 0;
      nsiz := blockettes_per_4kw;
      nrld := rdb.gid;
      noff := 0;
      nlen := 0;
      nsts := [ws$wrt];
      nsrb := 0;
    END;
  crw(wdb, status);
  IF status < > 1 THEN
    writeln('Can''t create address window. Status = ', status: 1, ' . ');
    current_4kw_block := -1;
  END; { Vinit }

FUNCTION vseek(record_number: word): data_pointer;
EXTERNAL;

FUNCTION vseek;

VAR
  needed_4kw_block: word; { Block containing desired record }
  offset_in_4kw_block: word; { Position of record in block }
  status: directive_status; { Status of MAP directive }

```

```

BEGIN { Vseek }
  record_number := record_number - 1; { Make relative to zero }
  IF (record_number < 0) OR (record_number > max_record) THEN
    BEGIN
      writeln('Virtual record number ', record_number + 1: 1,
        ' is out of range');
      needed_4kw_block := 0;
      offset_in_4kw_block := 0;
    END
  ELSE
    BEGIN
      needed_4kw_block := record_number DIV records_per_4kw;
      offset_in_4kw_block := (record_number MOD records_per_4kw) *
        data_size;
    END;
  IF needed_4kw_block < > current_4kw_block THEN
    BEGIN { Map new block containing desired record }
      wdb.noff := needed_4kw_block * blockettes_per_4kw;
      wdb.nlen := 0;
      map(wdb, status);
      IF status < > 1 THEN
        writeln('Can''t map to record ', record_number: 1, ' . Status = ',
          status: 1, ' . ');
      ELSE current_4kw_block := needed_4kw_block;
    END;
    vseek := loophole(data_pointer, wdb.nbas + offset_in_4kw_block);
  END; { Vseek }

PROCEDURE vget(record_number: word;
  VAR d: data);

EXTERNAL;

PROCEDURE vget;

VAR
  p: data_pointer;

BEGIN { Vget }
  p := vseek(record_number);
  d := p^;
END; { Vget }

PROCEDURE vput(record_number: word;
  VAR d: data);

EXTERNAL;
PROCEDURE vput;

VAR p: data_pointer;

BEGIN { Vput }
  p := vseek(record_number);
  p^ := d;
END; { Vput }

```

You can prevent Pascal tasks from using additional virtual memory by means of a global patch. The global variable `P$NEXT` can be patched to contain a value of 240 to prevent Pascal from extending into additional virtual addresses, as shown:

```
> TKB
TKB > TEST/FP/CP = TEST,VIRT,
TKB > LB:[1,1]PASLIB/LB
TKB > /
ENTER OPTIONS:
TKB > GBLPAT = TEST:P$NEXT:240
TKB > EXTSTCT = $$HEAP:20000
TKB > //
```

In earlier versions of Pascal this symbol is named `$NOEXT`.

When you prevent Pascal from using additional memory, you must provide explicit dynamic memory for the program by extending the heap via the `EXTSTCT` option as shown in the previous task-build example. For each program, the amount of memory required for heap storage varies according to the amount of local variable storage and the number of files your program opens. If your program dies with a stack overflow or with a not enough memory error, you should increase the size of the heap.

With the patch just described, your Pascal program will never use any additional virtual memory. This gives you complete control over the allocation of virtual memory within the task.

One of the principal advantages gained by using virtual memory is speed. In the case of the virtual file system described here, records are read and written by copying the data from the task's memory into the memory of a dynamic region. Records are accessed by moving a region's mapping window instead of by reading a disk file. This memory-to-memory copy operation is very efficient and can be done in a matter of micro-seconds rather than milli-seconds.

*Steve Poulsen is currently manager of Oregon Software's technical sales support. As one of the founders of Oregon Software, Steve has contributed heavily over the years to the company's development effort.*

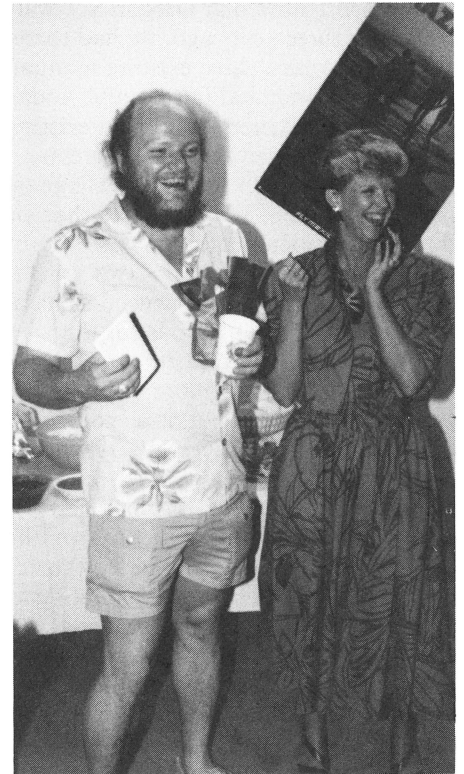
## Rites of Spring . . .

# Party Thrives in New Setting

Oregon Software's annual corporate party on May 17 moved indoors for the first time this year. Previously, we had always celebrated the company's incorporation at our old building on Canyon Road, where ample grounds and a secluded street made the commemoration a unique and well-attended event within the community. There were skeptics among those attending past celebrations who believed that the flavor of the old days would be lost in our new facility.

Like a die-hard C programmer being explained the benefits of Pascal, the skeptics were certain that such a conversion could never be made. What was going to happen to the bountiful barrels of imported beer that graced the halls of the previous parties? Probably replaced by wine coolers. What about the live folk music of the olden days? Probably done in by piped-in music from the dentist office down the street. Doubters were quickly reassured, however, as the party's featured event, Oregon Software's first-ever "Chili Cook-Off," immediately set the tone for much of the evening's activities.

An entire validation suite of tests was run on those entries submitted. The judges savored each dish and carefully cleansed their palates with Dos XX beer before moving on to the next entry. The dishes were tested for hotness, greasiness, viscosity, taste, and originality. After a long discussion, the judges decided that Bob Phillips, scoring 94 out of a possible 100 points, had won. The competition was extremely close, but Bob's chili had the robustness of a Pascal—2 compiler, and that in the end made the difference.



*Competition among the finalists in Oregon Software's first Chili Cook-Off was close, but Bob Phillips's "Rump-pot" entry was named the contest's winner.*

First prize was a night for two at Neskowin Resort on the Oregon coast.

After the judging, party-goers were invited to decide for themselves which of the chilies they preferred, as they drank margaritas and listened to mandolin folk music performed by Sandy Profeta and Jan DeWeese.

## RSTS/E Version 9 User's Note

We recently received our copy of the latest RSTS/E release from Digital Equipment Co. and have begun the process of testing and converting Pascal—2 to run under Version 9 of the operating system. We will announce the release of any new software required to run on Ver-

sion 9 as soon as it is ready for distribution. Customers with current support contracts who are upgrading to Version 9 can request the new release of Pascal—2 as soon as our announcement of its availability is made.

# Publication list grows as product list expands

by David Spencer

When I started at Oregon Software a little over three years ago, we had three technical writers, three existing manuals, and one new manual in progress. Today, we have three writers, seventeen existing manuals, and a new one in progress. Maintaining and improving our manuals demands more of our resources than producing new documentation.

A case in point: About two years ago, we announced that new editions of the Pascal—1 user manuals for the PDP-11 products were forthcoming (*Pascal Newsletter* Number 6, Spring 1983). What started out as a six-month project took a lot longer. While working on the PDP-11 books, we created manuals for Pascal—2 on three new microprocessors, for SourceTools on three DEC operating systems, and for the multi-optional cross-development system on two hosts. With the publication of the Pascal—1 V1.3 manuals in April, we've fulfilled our two-year-old promise and completed basic documentation on all current products.

Despite staff turnover and size limitations, the Technical Publications group plans to improve and extend those manuals as part of its routine maintenance. New sections on the support library, on floating-point format, and our implementation of Pascal are already in progress. We're experimenting with larger typefaces and more white space in our page layouts. Late in the summer, we will begin another cycle of comprehensive revisions.

As we plan the next round of

upgrades, we're looking through our files of users' suggestions and the feedback from our distributors' workshops. We're coordinating our efforts with the newly formed support group to gather further indications of your needs. We're developing better methods of production and

graphic presentation, in order to make our manuals easier to read and use. We need your feedback on our manuals to continue improving our product documentation. This is an opportune time to send us that suggestion you've always thought you'd make.

*OREGON SOFTWARE USER MANUALS: When we last printed a list of manuals and prices in this newsletter, we listed only Pascal—1, Pascal—2, SourceTools, Sort-1-Plus, and the concurrent package. We are again publishing a complete list of user manuals available. Prices and ordering information are provided in the Oregon Software Catalog of Products and Price Schedule (February 1985).*

## Oregon Software User Manuals

| Title                                    | Publication Date |
|------------------------------------------|------------------|
| Pascal—2 V2.1 for RSX                    | July. 83*        |
| Pascal—2 V2.1 for RSTS                   | Aug. 83*         |
| Pascal—2 V2.1 for RT-11                  | Aug. 83*         |
| Pascal—2 V2.1 for VMS                    | Oct. 84*         |
| Pascal—2 V2.1 for VAX/UNIX               | May 85           |
| Pascal—2 V2.1 for UNIX/68000             | Aug. 83          |
| Pascal—2 V2.1 for RT-11/Pro300           | Feb. 85          |
| Pascal—2 V2.1 for POS/Pro300             | Feb. 85**        |
| SourceTools V1.0 for VMS, RSX, RSTS      | Apr. 85          |
| Pascal—2 V2.0 for VERSAdos               | Nov. 82*         |
| Pascal—2 V2.0 Cross-Development RSX, VMS | Mar. 84*         |
| Oregon Linker/Assembler RSX, VMS         | May 85           |
| Pascal—2 Concurrent Programming Pkg.     | Apr. 84*         |
| Pascal—1 V1.3 for RSX                    | Mar. 85          |
| Pascal—1 V1.3 for RSTS                   | Mar. 85          |
| Pascal—1 V1.3 for RT-11                  | Mar. 85          |
| Sort-1-Plus V1.7                         | Apr. 80          |

\* Items marked with an asterisk are scheduled for an update within the next three months.

\*\* Abbreviated manual.

## Pascal—2 compiler available on ULTRIX

Oregon Software's Pascal—2 compiler now runs on ULTRIX-32, Digital's version of the UNIX operating system for VAX and MicroVAX computers, and on Berkeley UNIX (BSD4.2), from which ULTRIX is derived.

As with all Pascal—2 releases on Digital systems, Pascal—2 on ULTRIX-32 includes a high-level interactive debugger, an execution profiler and other utilities that aid in coding and development.

Pascal—2 conforms to the interna-

tional Pascal standard (ISO 7185) and produces extremely concise, fast object code, as compared to that produced by languages such as FORTRAN-77 and C. Pascal—2 supports all features of standard Pascal, including packed data structures and set types of up to 256 elements. The compiler allows calls to separately compiled routines written in Pascal, C, FORTRAN or MACRO, giving developers access to existing software libraries. Compile-time error-checking ensures data type conformance to the

Pascal standard, locates many uninitialized variables and detects nearly 150 Pascal syntax errors.

ULTRIX-32 is a licensed version of Berkeley UNIX that supports the entire BSD4.2 command set, including virtual memory demand paging. Different versions of ULTRIX-32 run on MicroVAX and VAX-11/700 Series computers.

Pascal—2 for VAX/ULTRIX is now available from Oregon Software or its authorized distributors.



## Book Review

by David M. Barnes

David M. Chess, *Programming in IBM PC DOS Pascal*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1985.

*Programming in IBM PC DOS Pascal* is written for the Pascal programmer developing applications on the IBM PC. The language described is PC DOS Pascal, versions 1.0 and 2.0, which is sometimes referred to in the text as "Pascal 2," by the way. Don't be fooled!

Because Pascal is an excellent teaching language, many Pascal books take the form of programming tutorials. In *Programming in IBM PC DOS Pascal*, the reader is referred to a standard introductory Pascal text for language details. This allows the author, who seems to be in touch with the needs of the applications programmer, to concentrate on the extensions of this dialect of Pascal and its interactions with the PC DOS operating system.

The book is divided into three parts. Part One is "Essential Pascal." It deals with the language itself, introducing the extensions peculiar to this dialect. The author accurately describes this part of the book as a "whirlwind tour." Part Two is "The Details," about the same length as part one, and deals with data files, screen handling, keyboard I/O and other DOS services, such as the assembler interface. This is the main part of the book. While it may slight some programmers' pet topics, it covers many important things that a developer needs to know. Part Three is built around a small programming project that ties in some of the earlier examples in the book. Mr. Chess also touches on some of the mandatory Pascal subjects: style, modularity, and top-down design. On these subjects he is refreshingly un-dogmatic.

PC DOS Pascal has many extensions to Standard Pascal, such as conditional compilation, loop terminators, separate compilation facilities (including a Modula-2 look-alike called a "unit"), an optional pseudo-code listing, and a variety of additional data types. Also provided are built-in functions that access PC DOS

system services.

Differences between the two versions are noted where they're important. Also noted, most of the time, are deviations from standard Pascal. Because the language's extensions are in large part the subject of the book, many deviations go unremarked. The author spends some time educating the reader in the uses of DOS, as well as Pascal, though without dissecting the operating system on an assembler level. Little time is spent on running the compiler (presumably covered in the manual) and the linker (also covered elsewhere).

The presentation of the book is good. It's slim, can be made to lie flat. Pascal keywords occurring in the text are

good example of the author's approach. Mr. Chess sets forth some reasons for devoting attention to the manner in which a program displays its output, then launches into describing four methods for doing so. These are: standard Pascal readln and writeln statements, ANSI.SYS calls, BIOS calls, and writing directly to screen memory. Each section is illustrated with specific examples. Finally, he offers "Some Speed Considerations," in which he mentions the tradeoffs involved in selecting one method over another. The quantity of technical material presented is surprising, especially considering that the chapter begins with the sentence "The display screen is the PC's voice."

The goal of the book, though the

---

### The book's strength lies in how well its author anticipates the readers' interests

---

printed in typewriter font so they can be picked out quickly when scanning. There is a summary at the top of each chapter, and the author encourages the reader to skip chapters in which the material seems too familiar.

Since all of the information exists in other forms, the strength of the book lies in how well its author anticipates the interests of his readers. I think he's done a remarkable job. The emphasis is on pragmatic information. The "Assembler Interface" chapter, for example, does not attempt to teach the reader PC assembly language, but it does provide the simplest possible assembler routine callable from Pascal.

The book's weakness is that it is (necessarily) sketchy in some areas that may be of interest to many programmers, and the level of understanding at which the book aims doesn't seem consistent throughout. There is sometimes a remarkable lack of precision in order to accommodate the novice reader, as in this description of `char` type variables:

"Char variables ... take values which are letters, punctuation marks, digits, spaces, and in general the sort of thing that one writes down"

The screen handling chapter is a

author didn't state it this way, seems to be to get the Pascal applications programmer up and running on his IBM PC as quickly as possible. The reader, in theory, has access to all of this material in the form of several separate manuals for the compiler, the linker, and the operating system. Mr. Chess relates these various parts of the development environment to each other in ways that most programmers will appreciate.

*David M. Barnes is a technical writer who came to Oregon Software via Compu-Serve Incorporated, where he developed interests in microcomputers and software tools.*

# Calling VMS system service and run-time library routines via Pascal—2

by Thomas E. Hanrahan

VMS system service and run-time library routines perform a wide variety of functions. They control resources available to processes, provide access to process information, permit communication between processes, manipulate character strings, and allow greater flexibility of input and output operations. Many of these procedures are useful programming tools and are available to Pascal programs.

Both the Pascal—2 compiler and support library running under VMS are written primarily in Pascal and rely on calls to VMS routines. One example is the support library procedure `p2_get_foreign`, used to read input directly from a command line. `P2_get_foreign` calls the VMS run-time library routine `lib$get_foreign` to return the contents of the foreign command line that invokes the compiler.

When you call a VMS system service or run-time library routine from a Pascal program, you must supply whatever arguments and calling sequence the utility requires. In all cases, VMS routines follow the standard VAX-11 conventions for calling procedures, so they must be declared as `nonpascal` when called from a Pascal program. To determine the arguments that a particular VMS routine requires, you must refer to the VAX/VMS manual in which the routine is documented, usually either the *VAX/VMS System Services Reference Manual* or *VAX-11 Run-Time Library Reference Manual*.

In the case of `lib$get_foreign`, the VMS run-time library routine follows the format:

where:

- `get-str` is a string to receive the text of the command line.
- `user-prompt` is a string to be used as a prompt for additional input if no command-line text is available.

- `rtn-len` is an integer variable passed by reference to return the length of the command line string.
- `force-prompt` is an integer variable set equal to 1 or 0 according to whether you want a prompt issued in all cases or only when command-line text is unavailable.

String arguments are generally passed to VMS routines by reference to VMS "string descriptors." Such string descriptors consist of four components:

- A word containing the length of the string.
- A byte specifying the data type of the argument.
- A byte specifying the class of the argument.
- Four bytes (described by DEC as a "longword") containing the address of the string.

Data-type and class values can be selected from tables provided in the *VAX-11 Run-Time Library User's Guide*.

A VMS string descriptor is defined in Pascal as a packed record whose fields correspond to the descriptor components just described. You begin setting up a string descriptor by defining the string buffer that the descriptor is to reference as some packed array of `char`. Next, you define two integer-subrange types that fit exactly into a byte (0..255) and a word (0..65535). And finally you define the descriptor itself, as shown in the type declaration of the example on the following page. Notice that pointers generated by the Pascal—2 compiler, such as `^cmdbuffer` in the example, are always four bytes in length and may serve as the fourth component of a string descriptor.

VMS system service and run-time library routines may be defined as either procedures or functions, depending on whether or not you want to test for successful completion of the called routine. When declared as a function, the routine returns an integer value. An odd return value indicates successful or non-

fatal completion.

In `p2_get_foreign`, the VMS routine `lib$get_foreign` is defined as a function with four parameters corresponding to the four required arguments for the routine.

The body of the procedure `p2_get_foreign` (not shown) is devoted to assigning appropriate values to the arguments that must be passed to `lib$get_foreign`. For instance, the descriptor `txt_descr`, references the string that returns the command line text, as follows:

```
with txt_descr do
begin
  len := h2-l2 + 1;
  class = 1; { fixed-length string }
  dtype := 0;
  txt_ref := ref (txt_buf);
end;
```

The descriptor `prompt_descr` is similarly assigned values. The string `prompt_buf`, which `prompt_descr` describes, is assigned the character string passed to `p2_get_foreign` from the calling procedure or main program. And the variable `force` is set to 0 because `p2_get_foreign` returns a prompt only when command-line text is unavailable.

Once assignments have been made to the various descriptors and variables, `lib$get_foreign` is called by the statement shown in the example. The routine stores the text taken from the command line in the string `txt_buf`. The contents of `txt_buf` can then be copied to the string `txt` by `p2_get_foreign` and subsequently passed back to the main program or calling procedure. There, the text can be parsed and processed as needed.

The usefulness of the procedure `p2_get_foreign` is not limited to passing command line text to the Pascal—2 compiler. As part of the Pascal support library, `p2_get_foreign` can be called from any Pascal program running under VMS as a foreign command. (See the section entitled "Reading Command Lines" in the recently released Update Package No.1 for Pascal—2 on VMS for further details.)

The ability of Pascal—2 to create an interface such as this one between p2\_get\_foreign and lib\$get\_foreign

is an important feature of the compiler, allowing you access to the full capabilities

of the VAX/VMS programming environment.

```

const
    cmdlinelength = 512;

type
    cmdindex = 1..cmdlinelength;
    cmdbuffer = packed array [cmdindex] of char;
    byte = 0..255;
    word = 0..65535;
    descriptor = _____ VMS string descriptor
        packed record
            len: word;
            dtype: byte;
            class: byte;
            txt_ref: ^cmdbuffer; _____ pointer filling a longword
        end;

function lib$get_foreign(var line1: descriptor;
                        prompt: descriptor;
                        var length1: word;
                        var forceprompt: integer): integer;
    nonpascal; _____ creates correct calling sequence

procedure P2_get_foreign(prompt: packed array [1..h1: integer] of char;
                        var txt: packed array [12..h2: integer] of char;
                        var length: integer);

    external;

procedure P2_get_foreign;

var
    ret_status: integer;
    txt_descr: descriptor;
    txt_buf: cmdbuffer;
    txt_length: word;
    prompt_descr: descriptor;
    prompt_buf: cmdbuffer;
    i, k: integer;
    force: integer;

begin
    :
    ret_status := lib$get_foreign(txt_descr, prompt_descr, txt_length, force);
    :
end;
```

*This code segment from the Pascal support library routine p2\_get\_foreign shows in Pascal terms the definition of a VMS string descriptor and a call to the VMS run-time library routine lib\$get\_foreign.*

# Errors, additions to manuals

Roughly every six months, we distribute update packages along with the release of new versions of our software. These update packages are made up of "change pages" documenting changes and additions to be included in each manual. In some cases, the changes describe new features or enhancements implemented in the software. Other times, the emendations are based on information or requests that we have received from readers through Documentation Evaluation Reports (the DER forms at the back of each manual), Trouble Reports, and support calls. (See the list of update packages issued to date.)

The cycle of update packages is staggered and generally depends on a product's original release date. During the interim between updates, we list changes and additions in this newsletter and the Release Notes. Note: whenever a change is made in response to a Trouble Report, we've placed the Trouble Report number in square brackets at the end of the entry.

## All Pascal—1 manuals

Our April release of the 1.3D Pascal—1 software coincided with our distribution of the 3rd edition of the Pascal—1 User Manual. Like the 1.3D software, the Pascal—1 User Manual has become an unsupported product. Whenever possible, however, we will publish corrections or additions to the manual in the *Pascal Newsletter*, and we encourage Pascal—1 users to send us tips or descriptions of problems that we may pass along to other Newsletter readers.

## All Pascal—2 manuals

Page numbers are not provided in this section because they vary from book to book. Instead, the location for changes is described by section name and paragraph or line number where appropriate.

## Oregon Software Update Packages

*Update packages are issued at roughly every other release of the software, with changes to intermediate releases being documented by Release Notes and this newsletter. Note: not all products are initially released as version A; Pascal-2 for VMS, for example, began as version 2.1C.*

| User Manual                           | Publication Date |
|---------------------------------------|------------------|
| <u>Pascal—2 V2.1 for RSX</u>          | July 1983        |
| Update Package No.1 V2.1B             | October 1983     |
| Update Package No.2 V2.1D             | February 1985    |
| <u>Pascal—2 V2.1 for RSTS</u>         | August 1983      |
| Update Package No.1 V2.1B             | October 1983     |
| Update Package No.2 V2.1D             | February 1985    |
| <u>Pascal—2 V2.1 for RT</u>           | August 1983      |
| Update Package No.1 V2.1B             | October 1983     |
| Update Package No.2 V2.1D             | February 1985    |
| <u>Pascal—2 V2.1 for VMS</u>          | October 1984     |
| Update Package No.1 V2.1D             | May 1985         |
| <u>Pascal—2 V2.1 for UNIX (68000)</u> | August 1983      |
| Update Package No. 1 V2.1D            | April 1984       |

## Constants not included in "Dead Code" elimination

Under the description of "Dead Code Elimination" in the section "Compiler Optimization," add the following sentence to the second paragraph:

The compiler does, however, generate constants in the constants' psect for string literals that appear in write and writeln statements within a dead code region.

[2425]

## PASMAT won't format %include files

In the section "PASMAT: A Pascal—2 Formatter," add to the list "Limitations and Errors" the following item:

- PASMAT does not process %include files. Consequently,

if the A directive is specified, an identifier is determined by that of its initial occurrence in the file being formatted and not by its form in the %include file.

[2201]

## All Pascal—2 manuals for PDP-11 and Professional 300 Operating Systems

### Embedded switch limitation

In the "Embedded Switches" section of the "Programmer's Guide," add the following sentence to the beginning of the fifth paragraph:

You may use up to 25 embedded switches in a compilation unit.

[2760]



**Extended-range arithmetic**

Replace the first paragraph of the "Extended-Range Arithmetic" section of the "Language Specification," with the following text:

The normal range of integer variables in Pascal—2 is -32767..32767, but you may also declare variable types in the extended range of 0..65535. A variable with an upper limit greater than 32767 is called an extended-range or "unsigned" variable. Any integer value may be assigned to an unsigned variable and is converted as a bit pattern. If the value being assigned is negative, the error is not trapped at run time, since there is no way for the compiler to tell the difference between a negative "signed" value and an extended-range "unsigned" value. The same sort of implicit transformation is true when an extended-range value is assigned to an integer variable.

The arithmetic operations of addition, subtraction, and multiplication are always signed operations. They treat both signed and unsigned variables as though they were signed. The results for these operations on signed variables are always correct; results produced for unsigned variables are correct except in overflow conditions involving multiplication. Division and modulo are unsigned operations for dividends in the extended range, but they do not treat divisors greater than 32767 as unsigned values. Comparison operations are signed or unsigned according to variable type.

**Pascal—2 for RSX****Overlay structures, pages 2-18 and 2-19**

Replace the last sentence in the description of the `DEBUG2` structure with:

For programs using single-precision, you have to specify `SING`, a special variant of the `SINGLE` co-tree, designed for use with `DEBUG2`. `SING` accounts for the fact that the Debugger is itself a Pascal program compiled with double precision.

In the third overlay example on page 2-19, change the `SINGLE` co-tree to read `SING`. [2803]

**Stack overflows, page 2-31**

Add the following text immediately before the section entitled "The 'Space' Function."

**Caution**

Very large stack overflows, those that extend beyond the heap and into the program code sections of memory, have the potential for causing serious problems. In some cases it is possible for the error to prevent the appropriate error message from being printed or the program from properly terminating. In some rare instances, the condition can even lead to the disruption of the computer's operating system. It is the programmer's responsibility to avoid such excessively large overflows by controlling the size of local variables and value parameters passed to procedures.

**Error walkback with I & D space programs, page 2-42**

At the the bottom of the page, insert the following paragraph:

Error walkback cannot be used with Instruction & Data (I & D) space programs available on RSX-11M/PLUS. Normally (in non-I & D space programs), the walkback routine examines instructions to find special markers and pointers to data areas, which contain procedure names and statement locations. But, in I & D space programs, instructions and data are kept separate, and this, coupled with the fact that data sections can be overlaid, makes it difficult for the walkback routine to function properly. In some I & D space programs, the walkback routine reports addresses instead of procedure names, but in other instances, the walkback procedure can abort with an odd address error or memory management trap. You should generally compile all I & D space programs with the `/nowalkback` switch to avoid such problems.

[2516]

**Program example, page 2-59**

In the program example, change the statement:

```
Efn := 1; {use event flag}
```

to read:

```
Efn := (16*256) or 1
```

The new line checks for all 8 words in the Status Block. [2800]

**Debugging with I & D space programs, page 4-1**

At the the bottom of the page, insert the following text:

**Caution**

The Debugger cannot be used on I & D space programs. The separation of instructions and data generated by such programs makes it difficult for the Debugger to properly trace procedures and statements. (See "Run-Time Error Reporting" for more details.)

**Pascal—2 for RT****Command line error, page 1-3**

The command line for running the program `CUSTOM` is wrong. It should be changed to:

```
.R CUSTOM
```

**Foreground Operations, page 2-49**

The last sentence on the page should be changed to read:

The period after 1024 signifies a decimal value.

**Pascal—2 for UNIX (68000)**

A number of pages in the manual contain minor errors that can be best changed by penciling in the changes as described:

**Multiple embedded options, page 2-5**

Below the sample embedded directive lines:

```
{ $noindex, norange }
{ $noindex } { $norange }
```

Add the sentence: "A space between options in the first command line, i.e. {\$noindex, norange}, is not allowed and the compiler treats everything after the space as an ordinary comment."

#### **Selective debugging, pages 2-5 and 2-6**

In several places on these pages, the manual states incorrectly that the debugging option can be turned "on" or "off" for sections within a module. The correct explanation is that the embedded options \$debug and \$nodebug actually have no effect on the current version of the Debugger for UNIX/68000. Debugging can occur only for an entire compilation unit and may be invoked only by specifying the -debug option on the compilation command line.

#### **Removal of parameters from the stack, page 2-15**

The last sentence in the fourth paragraph should read, "It is the responsibility of the calling procedure to remove parameters from the stack."

#### **Storage allocation for packed record, page 2-17**

The last sentence in the paragraph should read, "beginning at bit 8 (most significant bit)."

#### **'Read' and 'Readln' procedures, page 3-15**

Replace the existing heading with 'Read' and 'Readln' Procedures and add the following text after the section's first paragraph:

For variables of type subrange, read and readln statements accept values for the variable outside the limits defined for the subrange and the error is not recognized until such values are actually used. Error checking at the time of input, for values outside the limits of a subrange, is the programmer's responsibility.

#### **Program listing, page 3-21**

The type declaration of Word is incorrect in the program listing. The correct limits should be defined as 0..4294967295 and the program produces different results from the ones

shown in the example.

#### **Process termination values and statements, page 4-2**

Add the following paragraph just before the final paragraph on the page:

A process termination (exit) value and statement is given by the Debugger each time a program terminates, whether termination is normal or the result of some error. Exit values are either 1 or 0. An exit value of 1 always indicates that termination is the result of a run-time error and is preceded by the appropriate run-time error message. An exit value of 0 indicates either normal termination or termination because of some error other than run-time, such as a bus error. Except for normal terminations, error messages are also given for terminations that yield exit values of 0.

#### **Passing parameters in the Debugger via Go, page 4-10**

Replace the existing description for the G: Go command with the following:

The G (Go) command begins executing the program at MAIN,1; this command may be used at any point to restart the program. See the example after the C command.

If you debug a program that requires files to be passed as arguments, such files can be passed as parameters (similar to those in a write statement) with the G( 'file') command. For example, suppose that the program test.pas requires that data.dat be passed at some point during program execution. Compile test.pas and invoke the Debugger with:

```
pc -debug -output test test.pas
pdb test
}
Use the G command to pass
data.dat by entering:
} g('data.dat')
```

Note that the argument is enclosed in single quotes ( ' '). Multiple arguments must be enclosed individually in single

quotes and separated by commas, as:

```
} g('arg1','arg2','arg3',...)
```

#### **'For' statement index entry is incorrect, page Index-2**

The correct page reference for for statements is 3-28.

#### **Pascal—2 for VERSAdos**

The manual, soon to be released as a new edition, is being thoroughly reformatted to conform to the style of our other Pascal—2 manuals. The text contains minor corrections and additions throughout. Appendices listing compilation and run-time errors will be brought up to date.

#### **Pascal—2 Cross-Development System**

An update package is being prepared that reflects the switch from the OASYS cross-linker/assembler to the Oregon Linker/Assembler. The update package will also include an index for the first time.

#### *Continued from Page 1*

and several cross-compiler configurations. We will provide more details later.

#### **Other developments**

Maintenance work is continuing on our other products. The latest update for our VAX/VMS native compiler, version 2.1D, was released in early June.

Technical problems in the first shipments of version 2.1D of Pascal—2 for the PDP-11 caused us to pull back that release. The problem has been corrected, and customers who received the faulty shipments have been updated. All shipments of Pascal—2 for RSTS, RSX, and RT-11 are now Pascal V2.1D. Work on 2.1E will begin this summer.

# The Log: errors, work-arounds, changes

This log describes significant changes in Oregon Software products. As a service to our users who have reported problems to us in the past, we have provided Trou-

ble Report numbers where possible. At the end of this section, we've also listed reported bugs that have been verified by us as being problems with the software.

Where possible, work-arounds for known problems are provided after the description of the error.

## All Pascal—1 compilers

The following is a list of problems corrected by the 1.3D release.

| <u>Number</u>                | <u>Description</u>                                                                                                                                                                                  |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 508<br>TR 1213            | The assignment of a value to a function identifier went undetected. Now, the compiler generates the error message <code>Illegal assignment</code> .                                                 |
| TR 845<br>TR 1357            | The compiler failed to detect an illegal expression containing two consecutive operators. Now, it issues the error message <code>Bad expression</code> .                                            |
| TR 897a                      | Pascal—1 incorrectly evaluated boolean operations on integers with magnitudes approaching <code>maxint</code> . Such expressions are now correctly evaluated.                                       |
| TR 899<br>TR 1262<br>TR 1993 | External procedures declared twice in the same program caused the compiler to crash. Now, the error message <code>Bad procedure name</code> is issued.                                              |
| TR 1149                      | Expressions involving combinations of <code>pred</code> , <code>ord</code> , and <code>succ</code> functions occasionally returned incorrect results. Such expressions are now correctly evaluated. |
| Unassigned                   | The <code>get</code> procedure no longer fails when accessing a file of type <code>text</code> .                                                                                                    |

## Pascal—2 for VMS

The following is a list of bugs fixed in version 2.1D.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                   |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 2513       | The Pascal—2 command line prompt displayed two different strings. A single carriage return generated the string <code>PAS&gt;</code> ; a second carriage return resulted in <code>MP2&gt;</code> . The string <code>PAS&gt;</code> is now displayed in all cases.    |
| TR 2826       | The installation command file, <code>INSTALL.COM</code> , failed to create the Pascal—2 help library when an existing library structure was not already in place. Documentation has been added to the Installation Guide describing steps to take in such instances. |
| Unassigned    | The I/O control switch <code>/temp</code> failed to deallocate space used by temporary files, even though it deleted the corresponding file directory entries. The <code>/temp</code> switch now correctly deletes temporary files.                                  |

## Set problems corrected

|                    |                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 2501<br>TR 2511 | Defining a set as a structured constant caused the compiler to abort with an access violation. The compiler now generates the correct syntax-error messages for such conditions. |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- TR 2527  
TR 2548  
TR 2549  
TR 2759
- Certain set instructions failed to produce the correct code and in some cases resulted in an access violation.
- The compiler failed to generate an error message when a variable was declared to be a set of an enumerated type consisting of more than 256 elements. Such set declarations now produce the compilation error message Set types must have base in the range 0..255.
- Set expressions with non-scalar elements, such as strings, caused the compiler to crash.

**Conformant array bugs fixed**

- TR 2465  
TR 2526  
TR 2574  
TR 2602
- Certain programs hung when a call was made from within a for loop to a procedure with a conformant array parameter. Such programs now execute correctly.
- Undefined variables passed to conformant array parameters of a procedure or function were not trapped by the compiler and resulted in access violations. Such variables are now properly labeled by the compiler as undefined identifiers.
- Passing a structured constant by reference to a conformant array parameter resulted in the error message Too many nodes in procedure. The compiler now generates the correct compilation error Variable name expected.

**Readin', writin', and . . .**

- TR 2555  
TR 2713b
- Integers written to a file of `real` were not converted to real number format. This led to a run-time error when an attempt was later made to read such files. The compiler now generates code that converts integers to reals when they are written to a file of `real`.
- Attempts to read the real number 0.0 caused programs to loop indefinitely. The compiler now generates code that correctly reads the value 0.0.

**. . . 'Rithmetic problems fixed**

- TR 2651  
TR 2720  
TR 2815  
Unassigned
- Incorrect values were returned for `mod` operations performed on negative numbers. `Mod` now generates results according to the Pascal standard.
- The absolute value function, `abs`, incorrectly used a source register for a destination register. As a result, a variable with a negative value was changed to positive when it was used as an argument of `abs`. The compiler now processes absolute values correctly.
- The `mod` function returned incorrect results for unsigned integers when both the dividend and divisor were greater than 2147483647.

**'Reset' and 'rewrite' bugs fixed**

- TR 2576  
TR 2616
- Certain attempts to `reset` the standard file input caused a run-time error. The statement `reset(input)` now performs correctly.
- Text was stored in a file improperly when an attempt was made to `rewrite` the file after it had already been opened for writing. The compiler now generates code that correctly resets a pointer to the file's buffer.

**Debugger problems corrected**

- TR 2684
- When a command line included both file extensions for output files and the `/debug` switch, as shown in the example:

```
$ pas2 main.obj,main.lis = main.pas/debug
```

the VAX Linker detected illegal record sequences in the resulting object file. The compiler now generates the correct object code in these cases.



|            |                                                                                                                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unassigned | Variables stored in registers by the compiler returned incorrect values when watched or written by the Debugger. The Debugger now handles variables stored in registers correctly.              |
| Unassigned | Variables in the last line of a program were not previously accessible to the Debugger. Such variables are now available.                                                                       |
| Unassigned | The "watch" Debugger command W could not be set before the first line of a program was executed. With this release, watched variables can be specified before you begin execution of a program. |

## Pascal—2 for UNIX (68000)

Version 2.1G corrects the following problems with the compiler.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                        |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 2291       | The system call getpid, a C function declared as nonpascal in a Pascal program and used to return a program's process ID number, caused the run-time error Illegal instruction (core dumped) whenever it was called from within a write or writeln statement. The system call now functions properly within write and writeln statements. |
| TR 2666       | Mod operations computed incorrect results when performed on variables passed as parameters to a procedure or function. Such operations now generate correct values.                                                                                                                                                                       |
| TR 2667       | The compiler interpreted file names to be all lower case letters when searching for %include files. As a result, lookup failed on %include files whose names contained upper cases letters, such as Test.pas. The compiler is now case sensitive in searches for %include file names and recognizes file names with upper case letters.   |

## Problems with arrays corrected

|                               |                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 1917                       | The compiler crashed whenever a variable of type subrange was used as the bound in an array declaration, as with the variable z in the example:<br><br><pre> type   a = 1..7;   x = 1..7;   y = 1..7;  var   z: a;   ex: array [x,y,z] of integer; </pre> The compiler now generates the correct compilation error message Bad constant in such cases.                                                  |
| TR 1961<br>TR 2172<br>TR 2531 | The run-time error message Integer overflow resulted from illegal type declarations of the form type T = array [1..const-1], where an expression rather than a constant was used to define the limit of an array. The error is now trapped during compilation and identified with the proper syntax error message: ',' expected.                                                                        |
| TR 2660                       | Certain Subscript out of bounds errors were incorrectly identified as variable subrange exceeded errors. The problem occurred when a variable of type subrange was used as the index of a packed array and a value was assigned to the variable that exceeded the bounds of the index but not the limits of the subrange. The correct Subscript out of bounds error message is now given in such cases. |
| TR 2679                       | Changing the value of a single element in a packed array that was a field of a packed record caused other elements of the array to be similarly modified. Such operations on packed arrays within a packed record are now performed properly.                                                                                                                                                           |

## Known Bugs

We haven't completed our work on all the Trouble Reports that we have received; the following is a description of

those problems we've verified. Except for the problems noted with the now frozen Pascal—1 compiler, the bugs listed here

are all scheduled for correction in subsequent releases.

### Pascal—1

The following is a list of bugs outstanding in 1.3D.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 465        | When calling FORTRAN library functions using the <code>fortran</code> procedure option and the <code>/X</code> compilation switch (double precision reals), the function return value is lost.    |
| TR 846        | Double-precision function calls occasionally leave the stack in an inconsistent state.                                                                                                            |
| TR 968        | IMP-processed code generates a Branch out of bounds error during assembly.                                                                                                                        |
| TR 1069       | For the RSTS compiler, the <code>reset</code> procedure uses an assigned account number as the default directory.                                                                                 |
| TR 1348       | Assignment from a function passed as a parameter to a procedure generates an incompatible type error.                                                                                             |
| TR 1552       | A recursive procedure with a procedure parameter does not compile.                                                                                                                                |
| TR 1811       | The compiler fails to detect the second occurrence of an illegal operand.                                                                                                                         |
| TR 2174       | The compiler occasionally fails to diagnose a procedure call containing an incorrect number of parameters.                                                                                        |
| TR 2267       | Some file handling operations ( <code>Close</code> , <code>Reset</code> , <code>Rewrite</code> ) may generate spurious code unless followed by a semicolon at their call sites.                   |
| TR 2330       | In a program with nested procedures, a <code>goto</code> from within a <code>for</code> loop in the innermost procedure occasionally fails to find its destination in the outer procedure.        |
| TR 2335       | When writing to a file from a subscripted variable, the subscript calculation is sometimes incorrect.                                                                                             |
| TR 2388       | When a variable of type <code>text</code> is assigned to another variable of type <code>text</code> the compiler occasionally fails with an odd address trap instead of issuing an error message. |

### Pascal—2 for VMS

The following are known problems in version 2.1D.

| <u>Number</u>       | <u>Description</u>                                                                                                                                                                                                          |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 2514b            | The compiler fails to return a file name when it reports a run-time I/O error.                                                                                                                                              |
| TR 2550             | In certain cases, run-time errors that should generate the message <code>Variable subrange exceeded</code> are flagged as <code>Array index out of bounds</code> .                                                          |
| TR 2610             | When specified on the compilation command line, an illegal file name with two consecutive "dots" (i.e., <code>TEST..PAS</code> ), causes the compiler to generate the run-time error message <code>Can't open file</code> . |
| TR 2473<br>TR 2514d | The compiler fails to terminate with a "severe error" status when it detects compilation errors.                                                                                                                            |

|                    |                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 2642            | In certain cases the compiler fails to recognize an Array index out of bounds condition when compiled with checking enabled. Compiling with the /nocheck switch causes the errors to be correctly identified.                                                                                                    |
| TR 2713a           | The function trunc does not check for extended-range values and converts all unsigned real numbers to signed integers.                                                                                                                                                                                           |
| TR 2744<br>TR 2837 | In certain instances the compiler generates incorrect code for empty case statements with constant case selectors.                                                                                                                                                                                               |
| TR 2768            | The Dynamic String Package procedure insert( ) fails to truncate the target string properly when <i>max-len</i> characters is exceeded. Such overflows result in the run-time error Array index out of bounds.                                                                                                   |
| TR 2776            | The run-time error message File not open is produced when you use a single readln statement to input a packed array of characters and an integer from a text file, as in:<br><br><pre>readln(text, str, i);</pre> <p>where text is a file of text, str is a packed array of characters, and i is an integer.</p> |
| TR 2829            | Write statements generate spurious carriage returns when escape sequences or VMS run-time library calls are used to format screen output.                                                                                                                                                                        |
| TR 2862            | Readln(i), where i is of type integer accepts all types of input. As a work-around, use read(i) followed by readln to input integers.                                                                                                                                                                            |
| TR 2863            | Lack of an argument for the support library routine iostatus generates the error message Too many nodes in the procedure rather than the correct error message pair '(' expected and ')' expected.                                                                                                               |
| TR 2870            | The compiler incorrectly parses packed set elements of packed records when the set contains more than 32 members.                                                                                                                                                                                                |
| Unassigned         | The Debugger prompt } is followed by a spurious carriage return, causing your input to be written one line below the level of the prompt. Consequently, a typical Debugger command appears on your screen as:<br><br><pre>}<br/>  B(MAIN,5)</pre>                                                                |

## Pascal—2 for UNIX(68000)

The following are known problems in version 2.1G.

| <u>Number</u>      | <u>Description</u>                                                                                                                                                                                                                                                                 |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TR 1961<br>TR 2241 | Incorrect code is generated with certain element assignments of packed arrays of packed records when the record fields involved are sets. Pascal—2's optimizations cause the resulting code to be generated with byte rather than word instructions and lead to incorrect results. |
| TR 2536            | Certain programs that attempt to write elements of arrays that are themselves packed arrays of char return the run-time error message Illegal instruction (core dumped).                                                                                                           |
| TR 2586            | A mismatch between a type definition and a structured constant definition using that type causes the compiler to crash and return the error message Illegal instruction (core dumped).                                                                                             |

- TR 2589      Invoking both of the compiler options `-debug` and `$nolist` during a single compilation causes the compiler to generate an incomplete listing file and the Debugger crashes.
- TR 2730      Specifying a pointer to a file variable in the argument of a `rewrite( )` statement causes a core dump upon termination of the program. The program compiles and runs correctly except for the termination error, and the `rewrite( )` statement is properly executed. The core dumped error does not occur when a file variable is passed directly to the `rewrite( )` statement. This error was first reported in Version 2.1E.

### Problems in the Debugger

- TR 2445      The Debugger does not have access to function values because the symbol table file currently only recognizes variable names.
- TR 2658a      The Debugger does not write correct values of constants.
- TR 2658b      The Debugger fails to return an error message when you attempt to assign a value to a constant. The assignment, however, is never made, and the value of the constant remains unchanged.
- TR 2731      When a run-time error is encountered in a program being debugged, the Debugger is not able to recover the stack frame from which the error occurred and, as a result, it loses access to the program's variables. This condition will be corrected by the introduction of a post-mortem analyzer in a future release. As a temporary work-around, when you encounter a run-time error during debugging, set a breakpoint at the statement in which the error occurs, then restart the program and run it to that breakpoint. Once you reach the statement, you can accurately probe variables immediately before the point at which the run-time error appears.
- TR 2732      In modules compiled with the `$own` option, Debugger commands taking an argument `true`, `false`, or `nil` are flagged as Unknown identifier errors when set within a stack-frame context.
- As a work-around, return to the main program, set the command you wish to use, and proceed back to the breakpoint you want to examine. You can accomplish this using the command line:

```
} e(1);H(true)
```

### Oregon Linker/Assembler

The Oregon Linker/Assembler has two outstanding bugs at this time.

| <u>Number</u> | <u>Description</u>                                                                                                                    |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Unassigned    | The linker map fails to display sections containing no code. Sections that only define storage declarations do not appear on the map. |
| Unassigned    | Extraneous spaces, special symbols such as <code>\$</code> , and version numbers on the command line are flagged as illegal.          |



# Newsletter Index

This index catalogs topics that have appeared in the *Pascal Newsletter*. Entries are listed by newsletter number (in bold) and page number. Readers' comments on the format as well as the contents of this index are welcome.

---

## A, B

---

active page registers (APRs),  
     controlled address ranges, **9:4**  
     savings with virtual overlays, **8:1**  
 asynchronous system traps (ASTs),  
     implementation using DoCmd, **4:15**  
     Pascal—1 example, **8:20**  
 book reviews,  
     *Introduction to Pascal*, **3:3**  
     *Programming in PC DOS Pascal*, **10:9**

---

## C

---

checkpointing, **1:7, 3:2**  
 cluster libraries, **8:1**  
 common blocks, **7:21**  
 Concurrent Programming Package, **7:8**

---

## D

---

default devices incorrect, **2:4**  
 detached tasks, **8:8**  
 device control registers, **8:7**  
 device drivers, **7:9**  
 device monitors, **7:9**  
 disk space problems, under RT/TSX, **3:16**

---

## E

---

embedded systems, **7:8**  
 EMT predefined procedure, **2:5, 3:6**  
 EXTKEY directive, **3:2**

---

## F

---

FCSRES, **8:1**  
 file,  
     efficient packing of data, **7:4**  
     looking on wrong device under RT/TSX,  
         **2:3**  
     out of memory under TSX, **1:7**  
     overflow error, **4:21**  
     random access, **6:8**  
     speeding up opening process, **5:5**  
     unformatted FORTRAN structure, **7:4**  
 file variable under RT/TSX, **4:18**

FMS-11, **8:20**

FORINI, initialization routine, **4:22**  
 for loop restriction, **3:7**

FORTRAN,

    called from Pascal—2, **4:21, 4:22**

    calling Pascal—1, **1:5**

    FORINI routine, **4:22**

    I/O restrictions, **4:21**

FORTRAN carriage control, **6:5**

forward directive, **3:6**

---

## G, H

---

getpos, *see* random access

global symbols, access from Pascal programs,  
     **7:7**

\$\$heap,

    cross-reference, **2:3**

    extension of, **3:2**

---

## I

---

I/O channel numbers, retrieval of, **4:18**

I/O control switches,

    enhancing terminal performance, **8:11**

    implementing single-character I/O under  
         RSX, **6:4**

    reading unformatted FORTRAN files, **7:5**

    using FORTRAN carriage control under  
         RSX, **6:5**

I/O devices, access from Pascal, **8:7**

I/O errors, **6:6**

    error trapping, **6:7**

    printing under RSX, **6:6**

    sayerr procedure, **6:7**

I/O page, **8:7**

integers,

    integers, extended range, **6:11**

        overflow/underflow, **3:6**

        undetected overflow, **3:7**

        unsigned, **6:11**

interactive I/O, **6:10**

interrupt service routines, **4:10**

    CPP, **7:8**

    WAITIO, **7:12**

    interrupt vector initialization, **4:10**

---

**K, L**


---

kernel,  
 primitives, 7:11  
 structure, 7:15  
 lazy I/O, 6:10  
 synchronization of I/O, 6:10  
 literal strings, use in structured constants, 5:8  
 loophole function, 2:4, 9:6

---

**M**


---

mapping virtual to physical addresses, 9:4  
 memory management hardware, 9:4  
 memory organization,  
   under Pascal—1, 7:13  
   under RSX, 3:1, 9:4  
   under RT-11, 4:7  
 menu-driven modeling, 3:7  
 monitor error messages, 1:6

---

**N**


---

networking program, 3:7  
 newstart routine, *see* User Service Routine (USR)  
 nluns, double definitions, 4:22  
 nonpascal directive, 4:22  
   fortran renamed nonpascal, 2:3  
 not enough memory,  
   under RSX, 3:1  
   under RT/TSX, 3:16

---

**O**


---

OPUS, 5:2  
   Communique, 5:2, 6:14, 7:7, 8:22, 9:3  
   library, 7:7  
   membership list, 9:17  
 Oregon Pascal User's Society, *see* OPUS  
 origin, 2:3, 2:4, 4:23  
   CPP, 7:9  
   restrictions, 4:23  
   use with common blocks, 8:21  
 overlays, 8:1  
   NODSK attribute, 5:7  
   under RSX, 3:3, 5:7  
   under RT/TSX, 5:4  
   virtual, 8:1  
 overprinting, *see* FORTRAN carriage control

---

**P, Q**


---

partitions, , 9:4  
   allocation example, 9:5  
   creation of, 9:4  
   for resident libraries, 8:1

PAR utility, 8:1, 8:7, 9:4

Pascal—1, 3:5

  called from FORTRAN, 1:5  
   comparison with Pascal—2, 2:3, 3:5  
   compiler crashes, 1:6  
   real-time facilities, 7:11  
   text formatting, 3:9  
   variable length strings, 3:11

Pascal—2, 3:1

  calling FORTRAN routines, 4:21, 4:22  
   comparison with Pascal—1, 2:3  
   summary of new features, 6:3

Pascal programming tools, 9:3

  MEASURE, 5:3

  PVS Quality Control Package, 9:3

  Standard Pascal Mode Implementation,  
     9:3

  Syntax Checker, 8:4

PASLIB, 3:2

PASRES, 8:1

PLAS directives, 2:3, 8:1

portable database system, 3:7

preventing task extension, 2:3

privileged tasks, 8:8

process-control systems, 8:17

process-monitor model, 7:9

PSECTS, 3:2

Quick Task Builder, 4:2

---

**R**


---

random access, to text files, 6:8

random number generator, 4:3

record locking under RT/TSX, 4:18

record management systems, 9:3

  Pascal Record Management (PRM-11)  
     9:3

resident libraries, 1:8, 8:1

RMS-11K, 4:2, 8:20

ROM applications, 4:7

  cross-reference, 5:8

---

**S**


---

sayerr procedure, *see* I/O errors

/seek, 3:6

semaphores, 7:12

SET/MAXEXT, 3:2

setpos, *see* random access

SETSIZ program, 3:16

shared date, 9:4

shared tasks, 9:4

single-character I/O, 6:4

SJ monitor, restrictions, 4:21

spawning sub-tasks, 4:15

SQUEEZE command, 3:16

stack, overflows, 3:2

stand alone programs, 4:7  
start,  
    failure under RSX, 1:7  
    failure under RT/TSX, 2:4, 3:16  
    use of \$START, 3:16  
static local variables, 2:8  
surveys  
    consumer survey, 5:11  
    Modula-2 interest, 9:16  
Syntax Checker, 8:4  
SYSLIB, 3:2  
systems calls,  
    from RSTS, 2:5  
    from RSX, 4:15, 8:19

---

**T**

---

Task-builder, 3:2, 9:6  
task extension, 3:2  
task image, reducing size, 5:7, 8:1  
terminal I/O, 6:4  
transfer address, *see* transfer symbol  
transfer symbol, address incorrect, 4:21  
    incorrect for RT/TSX, 2:4, 3:16

---

**U, V, X**

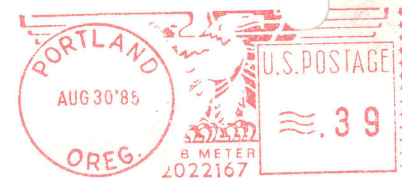
---

unsigned integers, *see* integers  
User Service Routine (USR),  
    reserving memory, 5:5  
variable length strings, *see* Pascal—1  
variables,  
    detecting initialized variables, 3:7  
    XREF utility, 3:7  
VMF utility, 9:5  
XM monitor, virtual jobs, 5:4  
XREF utility, 3:7

Pascal  
NEWSLETTER  
OREGON SOFTWARE

6915 S.W. Macadam Avenue  
Portland, Oregon 97219

John Peterson  
Apt #714  
130 South, 13th East  
Salt Lake City, UT, 84102





# Pascal NEWSLETTER

NUMBER 11

OREGON SOFTWARE

SPRING 1986

## Oregon Software Names New President

W. Walter Borowsky, who has more than 25 years' experience in marketing, operations, and computer engineering, has been named president and chief executive officer of Oregon Software.

Walt was president of Servio Logic Corporation of Portland from 1983 to

high-tech trading. He also had been director of applications software for CDC, professional services manager in Germany, and manager of international marketing in Europe.

Walt's hiring completed a two-year effort by Don and the previous management team to build Oregon

Software's business, marketing, and sales expertise. Walt's background in marketing and management is a very great asset. He led the growth of several business units within CDC that started smaller than OSI and grew to five or six times Oregon Software's size.



*W. Walter Borowsky*

1985 and was vice president and managing director of the China operations for Control Data Corporation (CDC) before that. Walt's experience at CDC covers 22 years in a variety of marketing, management, and engineering positions. He operated his own international consulting business after leaving Servio Logic.

Walt succeeds Don Baccus, one of Oregon Software's founders and one of the company's senior software engineers, who is returning to new product development.

Walt, who has a degree in electrical engineering, was responsible for all CDC activities in China when that country began to open up to outside

## Cross Development System leads new releases

Our December release of V2.0Q of the Pascal-2 development system completed months of intensive work on the VMS cross-compiler, RSX cross-compiler, VERSAdos native compiler, Oregon Linker/Assembler (OL/A), and Concurrent Programming Package (CPP) for the 68000.

We had actually field-tested the V2.0Q compilers months earlier, but the complete product package remained in field test as we resolved several difficult bugs relating to OL/A and CPP. Our final release of V2.0Q resolved the most serious outstanding problems in the 68000-related systems.

Once V2.0Q was completed, we began work immediately to merge the sources of the VMS-68000 cross-compiler, the VERSAdos native compiler, and the PDP-11 RSX native compiler into the common front end.

This work incorporates a number of bug fixes and product enhancements into the 68000 versions of Pascal-2 and simplifies future maintenance on all products.

We have frozen the RSX cross-compiler as of the 2.0Q release and will sell it only as an unsupported product in the future.

### OL/A Gains Speed, Power

Version 1.0C of the Oregon Linker/Assembler for VAX/VMS host systems is in field-test. Version 1.0C provides significant improvements in assembly and linkage times over Version 1.0B. The RSX-hosted version of OL/A will not migrate to version 1.0C; version 1.0B was the final release for OL/A under RSX.

*Continued on Page 6*

## In this issue . . .

|                                                               |         |
|---------------------------------------------------------------|---------|
| President's message . . . . .                                 | Page 2  |
| New staff introductions . . . . .                             | Page 2  |
| OPUS Communique . . . . .                                     | Page 3  |
| Information exchange . . . . .                                | Page 3  |
| Major benefits derived from host/target integration . . . . . | Page 4  |
| Field Report forms introduced . . . . .                       | Page 7  |
| Corrections, additions to the manuals . . . . .               | Page 8  |
| The Log . . . . .                                             | Page 12 |

## President's message

I am very excited to be part of Oregon Software, Inc. and recognize that we have a great technical team in place here in Portland.

A review of 1985 shows that the year was one of intense product development for Oregon Software. We brought out Pascal-2 on new systems and made our products available on various engineering workstations. We upgraded our cross-compiler products, including the release of our own linker/assembler, and made substantial headway on incorporating customers' suggested enhancements. We have streamlined our maintenance by developing a common front end to our Pascal-2 compiler.

The highlight of the year was the certification of a dozen Pascal-2 configurations as conforming to the U.S.

standards and the highest level of international standards available. This great achievement, which we believe is unparalleled in the industry, was topped off in February by the certification of an additional six compilers. This has been one of our main long-term goals and places us in the unique position in the industry of having certified products for VAX, PDP-11, Motorola (68000 family), Intel (iAPX 86 series), and National Semiconductor (32000 series) systems.

In summary, we got our product line in shape. In 1986, we will be concentrating on expanding our business. We want to become better known in our both our domestic and international markets. And we want to expand our activities to include high-level engineering consulting to specialized needs in the process control and automation industries. We want to improve our response to support calls and continue to improve the

quality of our products. As we concentrate on expanding our installed base, new and old customers will see an increased focus on productivity improvement tools and products. We have been receiving an excellent level of response to our recent user survey. We will be analyzing this data to provide better solutions to user problems and to position ourselves in the market for new products such as Modula-2.

We look forward to the challenge of 1986. I know that it will be an exciting year for us and our customers.



W. Walter Borowsky  
President, CEO  
Oregon Software, Inc.

## New talent added to staff

Veteran globe trotter **Hans Jonge Vos** joined our company last fall as leader of the Compiler Development Team. When **Bob Phillips** moved to Marketing, Hans took on the additional duties of Vice President of Engineering. Hans, who wrote his first cross-compiler in 1964, has twenty-six years' experience in computer software, bringing strength to his leadership position, particularly in the area of project planning. Hans is a Prairie Home Companion fan who also enjoys gardening, hiking and camping.



**Danna DeMarco** started as a Receptionist and, when **Katie Wilding** left for sunny Hawaii, Danna's sales experience made her a natural for the position of Assistant Buyer. Danna enjoys hiking and occasionally takes her two large dogs on ski trips.



As a Software Intern, **Laura Freeman** has taken on the difficult task of revamping our test suites. Laura is a Computer Science Major at Portland State University, where she also cultivates an interest in art history. In addition to her work on the test suites, Laura performs trial installations and recommends improvements to our installation procedures.



**Quan Dang** is a Software Intern who will soon graduate from Portland State University in Computer Science. Quan is a Vietnamese native who during his five years in this country has developed his taste for films and pop music. Quan performs final tests on release versions of the compilers, debuggers, and libraries.



If you have called with a support question recently, you have probably already met Support Engineer **Robert Pawelski**. In addition to answering phone calls, Robert is busy revising our support procedures, developing new formats for Field Reports and telex inquiries, and contributing to the design of our new customer database. Robert, who describes himself as "an idealist with one foot on the ground," is well-armed for support work with college degrees in both psychology and computer science. After work, he enjoys drawing with pen and ink, and reading Eastern history and philosophy.



*Continued on Page 7*



---

## OPUS Communiqué

*Oregon Pascal Users Society*

---

The Oregon Pascal Users Society needs your help! The recent resignation of Mr. Bruce Williams as coordinator has created the opportunity for you to get involved. Whether you can take over the lead or simply offer

a few minutes a week, we would like to hear from you.

A warm "thank you" is extended to everyone who stopped by to see us at DECUS/DEXPO in Anaheim, California in December. Special thanks to those of you who joined us at the OPUS Hospitality Suite. Our many conversations with users at DEXPO reinforced our sense of the need for and interest in an active users group.

Also, thanks to everyone who joined us at the first meeting of the Silicon Valley Users Group held January 17 at the San Jose Hyatt Hotel. The meeting not only let us look at users' needs and interests but

also offered attendees a look at Oregon Software's future plans and developments. The meeting also gave users in the Silicon Valley the chance to develop contacts with others doing similar things. The success of the breakfast encouraged more user activities in this area in the future!

This year promises to bring many new OPUS activities and services but the work cannot continue without your help and input. If you are interested in helping with the continuing efforts of OPUS, please write to Robert Pawelski at 6915 SW Macadam Ave., Portland, OR 97219 or phone him at (503)245-2202.

---

## Information Exchange

If you need information on technical applications involving Pascal, or if you have an application that might interest other users, send us a brief description for inclusion in the Information Exchange. Your description should follow the format of the item below. Interested parties can contact one another directly.

**PascEd** is a specialized tool for editing Pascal programs. It facilitates the job of Pascal programming

---

### **PascEd detects syntax errors**

because it views programs not as simple text but as Pascal structures that you can traverse and manipulate easily. Not only are whole structures inserted using very simple commands, but the editor maintains the format of the text, so no formatting characters need be typed.

PascEd performs all standard editor functions such as search, replace, cut and paste. Plus, it detects all syntax errors as they occur by checking text you enter or change. For more information on PascEd, write to RTP, Real Time Products, 1 Paul Street, London EC2A 4JJ, England.

---

## Pascal NEWSLETTER

OREGON SOFTWARE

6915 SW Macadam Avenue  
Portland, Oregon 97219

Thomas E. Hanrahan, editor  
David Spencer, David Barnes, Collins Hemingway, writers  
Betsy Slonaker, production assistant

The Pascal Newsletter is published regularly by Oregon Software, Inc., 6915 SW Macadam Ave., Portland, OR 97219; (503) 245-2202. Subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor. We pay \$100 per newsletter page for any article we print.

Copyright © 1986 by Oregon Software, Inc.  
ALL RIGHTS RESERVED.

Oregon Software, Pascal-1, Pascal-2, SourceTools, Oregon Linker/Assembler, and Pascal Newsletter are trademarks of Oregon Software, Inc. DEC, RSX, microRSX, RSTS, RT-11, P/OS, PDP-11, VMS, ULTRIX, VAX, and microVAX are trademarks of Digital Equipment Corporation. UNIX is a trademark of AT&T Bell Laboratories, Incorporated. M68000 and VERSAdos are trademarks of Motorola, Incorporated. XENIX and MS-DOS are registered trademarks of Microsoft Corporation. IBM is a registered trademark and PC-AT is a trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. HostStation is a trademark of Encore Computer Corporation.

Printed in USA

---

# Major benefits derived through integration on host/target systems

by Thomas E. Hanrahan

Any program must communicate with an operating system in order to run, but a program that is well integrated with an operating system exploits features of that system to run more efficiently.

By design, every Pascal-2 compiler is closely integrated with its host operating system. At the command level, the compiler conforms to native command structures and uses them to full advantage. At the language level, the compiler includes a flexible mechanism for calling system resources including utilities, libraries, and routines written in languages other than Pascal. And at the system level, Pascal-2 provides access to special low-level machine and operating system features that increase the compiler's control of the run-time environment.

Pascal-2 also lets you create programs that are well integrated with the target systems on which they run, whether your specific target is the host system itself (for native compilers) or some other system (for cross-compilers).

The benefits of host/target integration are substantial. Developers and their customers can rely on having a uniform environment in which to work; they need not learn new command structures to run the compiler or programs created with it. Programmers have access to a wide range of already existing resources that can speed the development of new programs. And thorough host/target integration promotes the creation of highly efficient programs.

## Uniform Development Environment

At the command level, the Pascal-2 compiler is designed to conform to the semantic and syntactic rules of the host operating system's command language. As a result, programmers who are accustomed to running native

assemblers or compilers find the commands for executing Pascal-2 quite natural.

Consider, for example, the family of Pascal-2 compilers that operate under UNIX. You run and control each compiler with the shell command `pc`, which is modeled after the `cc` command that invokes most native C compilers.

Like the `cc` command, `pc` accepts combinations of source and object files. (In the case of `pc`, source files may be written in Pascal or assembly language.) The `pc` command automatically routes source files to the compiler or assembler as needed.

## Programs run naturally on target systems

After compilation and assembly, the `pc` command sends all of the object files it has received to the loader, which generates the standard output file `a.out`.

Just as with the `cc` command, you can modify the compilation process by passing options to the `pc` shell script. Compare, for example, the C and Pascal command lines that you would use to compile a program, produce the additional files needed for running a debugger, and rename the executable output file:

```
cc -g -o tst tst.c
pc -debug -output tst tst.pas
```

The `-g` and `-debug` options in the two command lines trigger the creation of all necessary debugger files, while the `-o` and `-output` options direct the loader to rename the resulting executable files. In both cases, the output file is renamed `tst`.

Typical of UNIX command line semantics, Pascal-2 compilation options such as `-debug` are signified by

a hyphen followed by an option name. The two command lines shown here are indicative of the fact that Pascal option names are generally more descriptive than C option names.

In order to achieve a truly uniform command environment, both the compiler and the programs you produce with it must have access to native command structures. Such access allows you to write programs that run naturally on target systems. For example, if you were to write a spooler for a printer on a VAX/VMS system, you would want to run the software using Digital Command Language (DCL) syntax. A typical DCL command to execute your program might direct the printer to produce two copies of a file in landscape orientation:

```
spool sec.lst/cop = 2/land
```

The Pascal-2 compiler gives your programs access to those system service routines and system variables that store and return command-line text. For example, on VMS systems you can use the Pascal run-time library routine `p_get_foreign` to obtain the command-line text you need to control your program. In the case of the "spool" command line shown previously, `p_get_foreign` would return the string:

```
sec.lst/cop = 2/land
```

Once your program receives the command-line text from `p_get_foreign`, it can parse and interpret the information in whatever way you require.

`p_get_foreign` actually calls the VMS run-time library routine `lib$get_foreign` to retrieve the command-line text that the system stores. The Pascal routine supports all features of the VMS utility, but using `p_get_foreign` instead of `lib$get_foreign` saves you from having to define the various string



descriptors that the latter procedure requires.

For Pascal-2 compilers running under UNIX, the run-time support library contains the routine `u_getarg`. This function takes a numeric value, representing the position of an argument on the command line, and returns a "C-format" pointer to the location of that argument's string. You need only strip the UNIX null termination from the argument string to make it available to the rest of your program. With this feature, your programs can be executed just like any UNIX routine on your system.

### Expanded Resources

At the language level, Pascal-2 provides programmers with access to a wide range of resources that can greatly facilitate the creation of new programs.

### Programmers have access to many resources

The compiler offers two powerful directives, `external` and `nonpascal`, to let you call separately compiled or assembled routines from within a main Pascal program. You use the `external` label to identify routines compiled with Pascal-2 and the `nonpascal` label to identify routines generated by an assembler or some other compiler.

The `external` directive is especially useful for creating libraries of commonly used routines that you create with Pascal-2. If, for example, several applications require you to perform a lot of extended-range arithmetic, you can compile a separate module, accessible to all of your programs, that contains routines for handling output of unsigned integer and real numbers.

The `nonpascal` directive, on the other hand, gives you access to what is usually a large pool of already existing resources—not just procedures that you've written in other languages, but a variety of system services, utilities, and libraries. These resources often give you access to pro-

cess information and permit you to set up communications between processes. One specific example already mentioned for Pascal-2 on VMS is the utility `lib$get__foreign`, which the compiler calls (via `p_get__foreign`) to retrieve command-line text.

System utilities, such as `lib$get__foreign`, typically follow conventions for calling procedures that differ from those of Pascal-2, and you must identify any such routines as `nonpascal`. But once you declare a routine as `nonpascal` in your main program you need only supply whatever arguments the procedure or function requires when you make the call. Even such highly structured types as VMS string and item descriptors can be built in terms of simple Pascal types. [Ed. note—See Newsletter No. 10 for a detailed description of calling VMS system services via Pascal-2.]

More difficult but also possible is to call routines compiled by Pascal-2 from mainline programs written in other languages. We find this done most often on UNIX systems where Pascal subroutines are called from C programs, but such calls can be made from FORTRAN or assembly language programs on other systems as well.

The problems that you must resolve in order to call a Pascal subroutine from non-Pascal programs generally fall into two categories: initializing the Pascal run-time library without interfering with the run-time environment of the main program and establishing the correct calling protocol between the procedure and program. In some cases, you may also have to account for different machine instructions used by the Pascal and non-Pascal code to jump to subroutines. The process for making such calls is currently complex, especially if you wish to perform input or output operations in the Pascal routine, but our development plans include improvements in this area in the near future.

### Enhanced Program Performance

At the system level, the Pascal-2 compiler includes numerous extensions that facilitate programming by

providing access to special low-level machine and operating system features.

Loophole is typical of the Pascal-2 extensions that give you access to low-level features of the target computer system. The `loophole` function lets you escape Pascal's typing rules to coerce a variable of one type into another. For example, `loophole` permits you to convert a pointer to an integer in order to perform pointer arithmetic. Other extensions, such as `size` and `bitsize`, give you information about memory allocated by the compiler to different identifier types. And still others, such as `origin`, allow you to specify the contents of specific memory locations.

### Pascal-2 extensions facilitate programming

The most recent release of the VAX/VMS compiler (version 2.1E) demonstrates how you can take advantage of special operating system features to enhance a program's performance. In version 2.1E, our developers modified the way the compiler initializes the support library, altered the addressing mode used in a number of support library routines and rewrote sections of code to reference externally defined symbols. As a result both the compiler and support library can now be installed as "known images" on VMS.

Making an image "known" to VMS means that the operating system can locate the image file without a time-consuming directory search. In addition, you can assign the "shared" attribute to a known image, which allows multiple users concurrent access to major sections of the image. In other words, installing the compiler and support library as known, shared images speeds up the compilation process and eliminates the need for the system to create multiple copies of either image during execution.

### Ongoing Effort

Our plans call for Pascal-2 to be even more deeply integrated with the

systems on which the compiler runs.

At the command level, upcoming releases of Pascal-2 on UNIX will include a new procedure `getparam` that returns a Pascal-compatible command line argument to your program directly rather than a "C-format" pointer to a "C-format" string. `Getparam` will simplify the reading of command line arguments on UNIX systems. The next VAX/VMS release will enhance the compiler's use of DCL syntax. Among the changes will be an `/include` compilation switch that allows you to direct

the compiler to search specific user-defined directories for `%include` files.

At the language level, we are improving the mechanism for calling routines created by Pascal-2 compilers from non-Pascal main programs. Steps that are currently a programmer's responsibility, such as reversing the calling sequence of parameters, will be made transparent through extension of the `nonpascal` directive. And operations needed to initialize the Pascal run-time library and ensure the integrity of registers during proce-

dures calls will be greatly simplified.

Also at the language level, a new `descr` feature will allow the VAX/VMS compiler to automatically translate strings into VMS string descriptors when they are passed to VMS system services.

At the system level, we have already implemented a number of changes aimed at improving the speed of generated code. The main objective in all of these efforts is to make Pascal-2 and the programs it produces run efficiently and naturally on host and target systems alike.

### *Continued from Page 1*

Future development will augment the OL/A instruction set to support the M68020 processor and the M68881 floating-point coprocessor. We also plan to make OL/A available for other 32-bit operating systems.

### **Pascal-2 Comes to PC**

Another key development is Pascal-2 for the IBM PC-AT. Pascal-2 under XENIX went to field test in December and the MS-DOS version is expected this spring.

On other new high-performance systems, we have a preliminary implementation of Pascal-2 on the Encore HostStation, based on the NS32016, and we are now evaluating its performance.

### **MAKE Now Sold Separately**

MAKE, the automatic file rebuilder, is now available as a separate product as well as part of the SourceTools package. MAKE, which rebuilds software systems after individual components are changed, is available immediately on VMS and RSX systems. Pricing for RSX is \$995 plus \$200 for support. The VMS version is sold only by special quote and is not a standard product.

MAKE features several enhancements in its next scheduled VMS release (V1.1A). The new features include: support of long file specifications for VMS 4.0, including the dollar sign symbol in file names; a 30 percent improvement in speed; and the capability of MAKE to access the

creation date and time for an individual module of a library. We are considering similar changes for the RSX version of MAKE.

The RSTS version of the complete SourceTools package is now being sold as an unsupported product.

### **Pascal-1 Laid to Rest**

Finally, a note from the past: the last version of Pascal-1, V1.3D for RT-11, was released in December. The RSX and RSTS versions were released previously.

### **Product Schedule**

Recent and upcoming releases are on schedule (see inset). Schedules

cannot be guaranteed because a formal release date depends on product performance during internal testing and field testing. Customers should check with their sales representative as the dates approach to receive the most recent information. The published schedule will be revised each month as well.

### **Binders Changed**

Our user manuals are now being shipped with new easel-style binders. The new binder may be placed upright for easier reference when working at your keyboard. The binders also sport our new product color codes on their slip-in covers.

## **Recent and Upcoming Releases**

| Product                                | Field Test Release | General Release |
|----------------------------------------|--------------------|-----------------|
| Pascal-2 VAX/VMS to 68000 V2.0Q        |                    | Dec. 85         |
| Pascal-2 VAX/VMS V2.1E                 |                    | Jan. 86         |
| Pascal-2 VAX/UNIX V2.1E                |                    | Jan. 86         |
| Pascal-2 SUN/4.2 BSD V2.1I             |                    | Jan. 86         |
| Pascal-2 68000/VERSAdos V2.0Q          | Jan. 86            | Feb. 86         |
| Pascal-2 PDP-11/RSX to 68000 V2.0Q     | Dec. 85            | Feb. 86         |
| Pascal-2 IBM PC-AT XENIX V2.1D         | Dec. 85            | July 86         |
| Pascal-2 IBM PC-AT MS-DOS V2.1E        | Apr. 86            | July 86         |
| Pascal-2 PDP-11/RSX, RSTS, RT-11 V2.1E | May 86             | July 86         |
| Oregon Linker/Assembler V1.0C          | Mar. 86            | Apr. 86         |
| SourceTools VMS V1.1A                  | July 86            | Sept. 86        |
| SourceTools RSX V1.1A                  | Oct. 86            | Dec. 86         |

# Field Report forms introduced

In an effort to better serve our customers, we are continuing to develop and enhance our mechanism for processing Field Reports.

A new four-part Field Report form, previously called a Trouble Report, is being distributed to each customer who currently has a support contract with Oregon Software. (Contact your sales representative if you have questions regarding your support agreement.)

The Field Report form itself has undergone extensive modification and requires more information than

previous versions. The new information and detail that we are requesting will speed the processing of your request and contribute to the inherent quality of our product line by allowing us to better utilize the expertise and observations of our customers.

## Field Report Media Formats

When submitting Field Reports, you may find it necessary to send us program samples on floppy disk or magnetic tape. Be sure to label your media with the following

information:

- 1) Site #
- 2) Format
- 3) Density for floppies (except for RX50); BPI for tapes
- 4) Date generated

We can only accept a limited number of formats. The following table outlines the formats that we can read on various DEC operating systems. Customers sending files in other formats will be asked by mail to resubmit them in an acceptable format.

## Field Report Media Format

| Operating System   | Media      | Density         | Format       | Utility Used    |
|--------------------|------------|-----------------|--------------|-----------------|
| RT-11              | 8" floppy  | SS-SD or DS-DD  | RT-11        | PIP             |
| RT-11              | ½" magtape | 800 or 1600 BPI | RT-11 (ANSI) | PIP             |
| RSX-11M / RSX-11M+ | ½" magtape | 800 or 1600 BPI | DOS          | FLX             |
| RSTS               | 8" floppy  | SS-SD or DS-DD  | RT-11        | FIT             |
| RSTS               | ½" magtape | 800 or 1600 BPI | DOS          | PIP             |
| VMS                | ½" magtape | 800 or 1600 BPI | DOS          | FLX or EXCHANGE |
| P/OS               | RX50       | —               | Files-11     | COPY            |
| MicroVMS           | RX50       | —               | Files-11     | COPY            |
| MicroRSX           | RX50       | —               | Files-11     | COPY            |
| ULTRIX/UNIX        | ½" magtape | 1600 BPI        | TAR          | TAR             |

## Bits & Bytes

**Bits & Bytes** is a new feature of the newsletter, containing brief announcements and messages

### SourceTools generates undeleted disk files

In SourceTools V1.0C for VMS, we've discovered and corrected a bug that could have significantly affected your system's disk usage. Previously, SourceTools created directory entries for working files and deleted them when the job was done. However, the disk space allocated to those files was not returned and may still be unavailable to your system. To determine whether you have any "lost" files, run the DEC utility ANALYZE with the switches DISK\_STRUCTURE and REPAIR. This procedure gathers all lost files on your system disk and

places them in a directory where you may delete them.

### PAS.ODL on P/OS

The file PAS.ODL sent with Pascal-2 V2.1D for P/OS incorrectly references LB: [1,1] as the location of PASLIB. Most users have this in the directory [PASCAL2]. To use the ODL file, you must change all references to the appropriate directory.

### Update reminder

Oregon Software regularly distributes updated releases of its products. Such releases, however, are not sent to customers automatically and must be specifically requested for shipment.

## New Staff

*Continued from Page 2*

Leslie Lee was an Administrative Assistant for a major oil company when she decided to take some time out to have a baby. We're glad Leslie chose to rejoin the work force as a receptionist at Oregon Software. Like many native Oregonians, Leslie is an avid runner. She also enjoys restoring her turn-of-the-century house in southeast Portland.



# Corrections and additions to the manuals

Certain pages in our manuals contain minor errors or require simple additions that can be best changed by pasting or penciling in the corrections described below. Your notes will serve as a reminder of the changes until you

receive corrected pages in the form of an update package in the future. The cycle of update packages for Oregon Software products extends for roughly six months and depends on the ori-

ginal release date of the product. Note that complete revisions of the user manual for the PDP-11 systems (RSX, RSTS, and RT-11) will accompany the 2.1E release of the software.

## Pascal-2 for VAX/VMS

The following changes include supplemental material and errata.

### Page number

### Correction or addition

2-54

We have increased the maximum allowable number of certain Pascal symbols. Note the following new values in your manual:

Too many forward declarations (only 1023 allowed)

Too many identifiers (only 4999 allowed)

Too many procedures (only 1023 allowed)

3-17

Replace the heading that begins **String Input . . .** with **'Read' and 'Readln' Procedures** and after the section's first paragraph, add the following text:

"For variables of type subrange, read and readln statements accept values outside the limits defined for the subrange and the error is not recognized until such values are actually used. At the time of input, the programmer must check for values outside the limits of a subrange."

3-17

Add the new subsection:

#### **'Write' and 'Writeln' Procedures**

The Pascal standard allows you to write character strings of the form packed array [1 . . n] of char, where n is some integer greater than 1. With Pascal-2, you may also use write and writeln to write strings of the form packed array [1 . . 1] of char.

3-17

Add the new subsection:

#### **Real Number Formatting**

The default representation for real numbers is scientific notation with a field width of 13. (See "I/O Definitions" for details.) You may, however, use colon notation to alter the way in which real numbers are written. By specifying a single formatting field, you can control the field width of the printed number. The representation of the number remains in scientific notation but the number of digits displayed is determined by your field-width specification. To generate conventional decimal notation, you must add a second formatting field. When you use two formatting fields, the first value determines the overall field width of the number and the second value specifies the number of digits to follow the decimal point. (The standard allows you to specify an integer constant or an integer expression in either formatting field.)



For example, the following code segment and output correspond to a value of  $R$  equal to  $-367.2$ :

```

:
writeln('R = ',R);
writeln('R = ',R:10);
writeln('R = ',R:10:2);

```

```

:
```

```

R = -3.672000e+02 _____ default field width is 13
R = -3.672e+02
R =      -367.20

```

- 3-23 Two errors appear in the sample program. Immediately below the program statement `program Coerce` in the example, insert the line `{ $double }`. Then, in the statement that begins `writeln('Re = '...)`, change the integer field width specifications from `-6` to `-8`. The new program results differ from those given in the example.
- 3-24 Add the following sentence to the paragraph preceding the program example `MDump`:
- “Note also, that as with the function `ref`, pointers generated by `loophole` fail normal pointer access checks and the embedded option `$nopointercheck` must be specified whenever a pointer produced by `loophole` is used?”
- 3-26 Ignore the subsection entitled ‘**Mod**’ of **Negative Numbers**. The Pascal-2 compiler now implements modulo according to the Pascal standard.
- 4-13 Add the following sentence to the end of the next to last paragraph on the page:
- “The Debugger, however, ignores case variants in records and writes only the value of the first identifier in the case branch list.”

## Pascal-2 for VAX/UNIX

The following changes include supplemental material and errata.

| <u>Page number</u> | <u>Correction or addition</u>                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1-6                | Remove the line that reads, “The assembly and object options are mutually exclusive.”                                                                                                                                                                                                                                                           |
| 1-13               | Change the loader command line example to read:<br><pre>\$ ld -o execfile /lib/crt0.o srcfile.o \! /usr/lib/libp.a /usr/lib/libc.a</pre>                                                                                                                                                                                                        |
| 1-15               | Change the Debugger command line example to read:<br><pre>\$ pc -debug -output efact efact.pas</pre>                                                                                                                                                                                                                                            |
| 2-20               | At the end of the third paragraph, add the sentence: “The <code>%include</code> directive is case sensitive and recognizes file names with upper-case letters.”                                                                                                                                                                                 |
| 2-33               | In standard Pascal, upper-case and lower-case letters are interpreted the same; the Pascal-2 compiler translates all to lower case. Consequently, external procedure and function names that are called by a program must be all lower case. For clarity, change all examples of the routine <code>Cfunction</code> to <code>cfunction</code> . |

- 2-45 We have increased the maximum allowable number of certain Pascal symbols. Note the following new values in your manual:
- Too many forward declarations (only 1023 allowed)
  - Too many identifiers (only 4999 allowed)
  - Too many procedures (only 1023 allowed)
- 3-15 Replace the heading that begins **String Input . . .** with **'Read' and 'Readln' Procedures** and, after the section's first paragraph, add the text that appears in the change to page 3-17 of the Pascal-2 for VAX/VMS manual:
- 3-16 Add the same new subsection **'Write' and 'Writeln' Procedures** that appears as a correction to page 3-17 of the Pascal-2 VAX/VMS manual.
- 3-16 Add the same new subsection **Real Number Formatting** that appears as a correction to page 3-17 of the Pascal-2 VAX/VMS manual.
- 3-17 Note in the second to last paragraph on the page that the multiplication operation is not always signed but is signed or unsigned according to operand type.
- 3-22 Make identical changes to those described in the correction to page 3-23 of the Pascal-2 for VAX/VMS manual.
- 3-23 Make an identical change to that described in the correction to page 3-24 of the Pascal-2 for VAX/VMS manual.
- 3-25 Delete the subsection entitled **'Mod' of Negative Numbers**. The Pascal-2 compiler now implements modulo according to the Pascal standard.
- 4-17 Make an identical change to that described in the correction to page 4-13 of the Pascal-2 for VAX/VMS manual.

## Pascal-2 for M68000/UNIX

The following changes include supplemental material and errata.

| <u>Page number</u> | <u>Correction or addition</u>                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2-9                | At the end of the third paragraph, add the sentence: "The %include directive is case sensitive and recognizes file names with upper-case letters."                                                                                                                                                                                                                                         |
| 2-13               | In standard Pascal, upper-case and lower-case letters are interpreted the same; the Pascal-2 compiler translates all to lower case. Consequently, external procedure and function names that are called by a program must be all lower case. For clarity, change all examples of the routine Cfunction to cfunction.                                                                       |
| 2-21               | In the subsection entitled 'Constant Folding' remove <code>real</code> from the list of types that are folded during optimization. For customers with source releases on systems implementing IEEE floating point: Incorrect results can occur if you attempt to fold single-precision <code>real</code> values with a compiler that has been built with double-precision characteristics. |
| 2-31               | The limits on the maximum number of certain Pascal symbols have been increased. Note the following new values in your manual: <ul style="list-style-type: none"> <li>Too many forward declarations (only 1023 allowed)</li> <li>Too many identifiers (only 4999 allowed)</li> <li>Too many procedures (only 1023 allowed)</li> </ul>                                                       |
| 3-16               | Add the following sentence to the end of the section entitled "Hexadecimal Output":<br>"When writing the hexadecimal value of a negative integer, the number is printed with a field width of at least 8 to show sign extension in the 32-bit quantity?"                                                                                                                                   |

3-23

Add the two new following subsections:

#### Procedure 'Delete'

The predefined `delete` procedure allows the deletion of a single file that is opened in a Pascal program. `Delete` accepts one argument, the file variable of the file to be deleted. Invoke the procedure with a statement similar to the following:

```
delete(F);
```

Internally, this procedure closes and deletes the file specified by the argument. Your program should not close the file (using `close`) before invoking the `delete` procedure. The run-time error message "File not open" results if the file cannot be deleted for some reason.

#### Procedure 'Rename'

The predefined procedure `rename` allows the renaming of an open file, from within a Pascal program. `Rename` accepts two arguments. The first argument passes to `rename` the file variable of the original file name. The second argument specifies the new file name and may be a constant, variable, or literal string. Invoke the procedure with a statement similar to the following.

```
rename(F, 'newfile.txt'); — renames F to newfile.txt
```

or

```
NewF := 'newfile.txt'
rename (F, NewF)
```

The original file must be open (via `reset`) before `rename` may be called on the file. The renamed file is automatically closed upon completion of the operation.

4-17

Add the following sentence to the end of the penultimate paragraph on the page:

"The Debugger, however, ignores `case` variants in records and writes only the value of the first identifier in the `case` branch list."



Make it a Full  
House —  
JOIN US!

9th Annual  
Open House

OREGON  SOFTWARE

'An Evening  
in Monte Carlo'

Friday  
May 16, 1986  
4:30 pm to 7:30 pm

6915 S.W. Macadam Ave.  
Suite 200  
(503) 245-2202

# The Log: corrections, changes, and work-arounds

This log describes significant changes that Oregon Software has made to its products in response to reports and suggestions submitted by users. We have also reported bugs that

have been verified by us as being problems with the software but have yet to be corrected. Where possible, we provide work-arounds for known problems after a description of the error.

As a service those users who have reported problems to us, we have recorded Field Report numbers for applicable entries.

## Pascal-2 for RSX

The following problems are verified for V2.1D Pascal-2 running under RSX. Many of the bugs listed here have been fixed in Version 2.1E, which

was undergoing validation tests at the time this newsletter went to press. Quality assurance tests are expected to confirm correction of many more

problems. Field Reports prior to #3100 are not reported, but users have been notified of the report's status.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                     |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3149          | The installation command file PASRES.CMD contains instructions that do not work on RSX-11M-Plus. As a work-around, users may substitute the following instructions:<br><br><pre>SET/TOP = GEN: &lt;size&gt; SET/PAR = PASRES: &lt;base&gt; : &lt;size&gt; : SYS INS PASRES/RON = YES/PAR = PASRES FIX PASRES/REG</pre> |
| 3166          | Programs with packed structures may cause the compiler to fill the disk on which the work files reside then crash with seek to record zero when the disk is full.                                                                                                                                                      |
| 3177          | No documentation is given for compilation error 142.                                                                                                                                                                                                                                                                   |
| 3186          | Attempts to generate an absolute resident library may fail under I&D space separation. The user manual is unclear on the procedure.                                                                                                                                                                                    |

## Pascal-2 for RSTS

The following problem is verified for V2.1D Pascal-2 running under RSTS/E. The problem is fixed in the V2.1E release, which was undergoing

validation tests at the time this newsletter went to press. Quality assurance tests are expected to confirm an even greater number of bugs

fixed. The Version 2.1E *Release Notes* will provide a complete list of corrected and known problems.

| <u>Number</u> | <u>Description</u>                                                                                                                         |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 2839          | The compiler fails to display the contents of a real variable in a write statment when that write statment is the last one in the program. |



## Pascal-2 for RT

The following problems are verified for V2.1D Pascal-2 running under RT-11. Many of the bugs listed here have been fixed in Version 2.1E, which was undergoing validation tests at the

time this newsletter went to press. Quality assurance tests are expected to confirm correction of many more problems. With the exception of the Field Reports noted below, reports

prior to #3100 are not recorded, but users have been notified of the report's status.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                       |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2382          | Certain programs compile with errors under TSX but not under RT-11.                                                                                                                                                                                      |
| 2861          | Program dies when reading a text file that contains tabs and integers.                                                                                                                                                                                   |
| 2985          | Certain programs fail to compile under the SJ monitor but not under TSX.                                                                                                                                                                                 |
| 3173          | A program compiled with the /list switch on may enter an infinite loop at the end of the first listing pass, if the program contains more than 19 resetttings of the embedded \$list and \$nolist switches or if the compiler encounters a syntax error. |
| 3182          | Incorrect and out-of-range values may be returned because variables of type char or of type 0..255 are sign-extended when assigned to temporary registers.                                                                                               |

## Pascal-2 for P/OS

The following problem is verified for V2.1D Pascal-2 running running under P/OS. The problem will be corrected in the V2.1E release.

| <u>Number</u> | <u>Description</u>                                                                                                         |
|---------------|----------------------------------------------------------------------------------------------------------------------------|
| 2995          | The command string interpreter does not accept file names and switches entered on the same line as the invocation command. |
| 3073          |                                                                                                                            |

## Pascal-2 for VAX/VMS

Version 2.1E, released December 27, corrects the following problems.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                             |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2514b         | The compiler failed to return a file name when it reported an I/O error.                                                                                                                                                                       |
| 2550          | In certain cases, run-time errors that should have generated the message Variable subrange exceeded were reported as Array index out of bounds. The new error message Array index out of bounds or Variable subrange exceeded is now reported. |
| 2632          | An internal compiler error caused a program to produce an undeleted temps error during compilation. The program involved use of the mod operation and assignment of elements in an array of integers.                                          |
| 2646          | An inconsistency between the analys and travrs phases of the compiler has been resolved.                                                                                                                                                       |
| 2739          | Some programs displayed the real number output of zero as 0. <i>n</i> instead of 0.0, where <i>n</i> was some integer other than 0.                                                                                                            |

|            |                                                                                                                                                                                                                                                           |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2744       | In certain cases, the compiler generated incorrect code for empty case branches.                                                                                                                                                                          |
| 2837       |                                                                                                                                                                                                                                                           |
| 2776       | Using a single read or readln statement to input a packed array of characters and an integer from a text file generated the error File not open.                                                                                                          |
| 2795       | Directory entries for files created with the I/O control switch /temp were deleted upon a program's termination but the disk space allocated to those files was not returned to the operating system. Programs now correctly return such allocated space. |
| 2870       | The compiler incorrectly parsed packed set elements of packed records when the set contained more than 32 members.                                                                                                                                        |
| 2888       | The PASMACH chapter of the user manual included several incorrect examples and descriptions. Update Package No. 2 corrects all of the problems.                                                                                                           |
| 2903b      | In reset and rewrite statements a file name string padded with null characters produced a memory access violation.                                                                                                                                        |
| 3008       | The compiler generated incorrect byte and word alignment for certain packed records. The compiler now produces correct alignments.                                                                                                                        |
| 3027       | Control-C ( ^ C) failed to stop execution of the Debugger. Now, when you enter a single ^ C the Debugger stops executing. (This allows you to break out of infinite loops.) When you enter two consecutive ^ C commands, you quit a debugging session.    |
| 3049       | When rewriting standard output to a file, some write statements caused programs to send output to the terminal rather than the specified output file.                                                                                                     |
| Unassigned | The write command of a numeric constant, such as W(27740), returned the address (27740) rather than writing the contents of the address.                                                                                                                  |

## Known Problems

The following is a list of known problems in version 2.1E of Pascal-2

for VAX/VMS. All are planned for correction in a

future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                           |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2642          | In certain cases involving constant indexes, the compiler fails to recognize Array index out of bounds conditions when checking is enabled. Compiling with the /nocheck switch causes the errors to be correctly identified. |
| 2768          | The Dynamic String Package procedure insert( ) fails to truncate the target string properly when <i>max-len</i> characters is exceeded. Such overflows result in the error Array index out of bounds.                        |
| 2825          | The break procedure generates carriage returns and line feeds for variables of the type file of char.                                                                                                                        |
| 2829          | Write statements generate spurious carriage returns when escape sequences or VMS run-time library calls are used to format screen output.                                                                                    |
| 2862          | The predefined function ioerror does not recognize input errors for readln(i), where i is of type integer. As a work-around, whenever you perform tests with ioerror, use read(i) followed by readln to input integers.      |
| 2863          | Lack of an argument for the support library routine iostatus generates the error message Too many nodes in procedure rather than the correct error message pair '(' expected and ')' expected.                               |
| 2902          | In certain cases, the compiler enters an infinite loop when the keyword goto is misspelled as go to.                                                                                                                         |

- 2903a Attempts to input the real number —0.0 cause programs to fail with a reserved operand fault error.
- 2935 The predefined function `iostatus` produces incorrect error codes.
- 2953 Line numbers are sometimes reported incorrectly in the walkback when `%include`  
3019b files are used.
- 3002 An internal compiler error produces an undeleted temps error during compilation.
- 3003 In certain cases, leaving the single quotes out of a `%include` file name causes a program to fail with an access violation.
- 3004 No error message is reported when an otherwise legal comment is not closed at the end of a file.
- 3005 The compiler generates incorrect code when you attempt to assign values to an array whose elements are of type packed array [1..8] of char.
- 3019a A function whose return type is a packed array of characters generates incorrect stack offsets.
- 3132a Certain programs, in which a structured variable is declared with the origin statement, fail with an access violation.
- 3147 For a real variable declared with a fixed origin, the compiler incorrectly sets the address of the variable to zero.
- 3158 A certain syntactically incorrect program causes the compiler to fail with an access violation.
- 3161 Certain attempts to write to a file opened with `reset( )` can cause a program to fail with an access violation.
- 3168 In some cases, the compiler recomputes the address of a record even though the address is available in a register.
- 3172 You may find spurious text displayed when you attempt to run a program immediately after using Control-Y to interrupt another program that was transmitting output to the terminal.
- 3202 The compiler produces position dependent code when walkback is enabled.
- Unassigned Programs compiled with the `$nowalkback` embedded switch produce illegal object code that cannot be linked. As a temporary work-around, use the `/nowalkback` compilation switch to disable run-time error walkback reports.
- Unassigned In certain cases involving non-local `goto` statements, the compiler generates incorrect embedded code for the Debugger and Profiler. Programs compiled with the `debug` or `profile` switches and affected by the problem fail with the error message Fatal error— Stack underflow.
- Unassigned The Debugger prompt `}` is followed by a spurious carriage return, causing your input to be written one line below the level of the prompt. Consequently, a typical Debugger command appears on your screen as:

```
}  
B(MAIN,5)
```

## Pascal-2 for VAX/UNIX

Version 2.1E, released December 17,  
corrects the following problems.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                      |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2445          | The Debugger did not have access to function values because the symbol table file recognized only variable names. The symbol table file now recognizes both function and variable names.                                                                |
| 2550          | In certain cases, run-time errors that should have generated the message Variable subrange exceeded were reported as Subscript out of bounds. The new error message Array index out of bounds or Variable subrange exceeded is now reported.            |
| 2632          | An internal compiler error caused a program to produce an undeleted temps error during compilation. The program involved use of the mod operation and assignment of elements in an array of integers.                                                   |
| 2646          | An inconsistency between the analys and travrs phases of the compiler has been resolved.                                                                                                                                                                |
| 2739          | Some programs displayed the output of zero as 0. <i>n</i> instead of 0.0, where <i>n</i> was some integer other than 0.                                                                                                                                 |
| 2744<br>2837  | In certain cases, the compiler generated incorrect code for empty case branches.                                                                                                                                                                        |
| 2776          | Using a single readln statement to input a packed array of characters and an integer from a text file generated the error File not open.                                                                                                                |
| 3008          | The compiler generated incorrect byte and word alignment for certain packed records.                                                                                                                                                                    |
| 3027          | Control-C ( ^ C ) failed to stop execution of the Debugger. Now, when you enter a single ^ C the Debugger stops executing. (This allows you to break out of infinite loops.) When you enter two consecutive ^ C commands, you quit a debugging session. |
| Unassigned    | In some cases, the Debugger write command (W) failed to write the final field in a record.                                                                                                                                                              |
| Unassigned    | The makefile that we provided to install the Pascal-2 development system occasionally overwrote certain existing UNIX files. We have renamed several files to correct the problem.                                                                      |

## Known Problems

The following is a list of known  
problems in version 2.1E of Pascal-2

for VAX/UNIX. All are  
planned for correction

in a future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2642          | In certain cases the compiler fails to recognize Array index out of bounds conditions when compiled with checking enabled. Compiling with the /nocheck switch causes the errors to be correctly identified.                                                                                                                                                                                                                                                                                                                                                               |
| 2731          | When a run-time error is encountered in a program being debugged, the Debugger is not able to recover the stack frame from which the error occurred and, as a result, it loses access to the program's variables. This condition will be corrected by the introduction of a post-mortem analyzer in a future release.<br><br>As a temporary work-around, when you encounter a run-time error during debugging, set a breakpoint at the statement in which the error occurs, then restart the program and run it to that breakpoint. Once you reach the statement, you can |



|            |                                                                                                                                                                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | accurately probe variables immediately before the point at which the run-time error occurs.                                                                                                                                                                                                            |
| 2768       | The Dynamic String Package procedure <code>insert( )</code> fails to truncate the target string properly when <i>max-len</i> characters is exceeded. Such overflows result in the run-time error Array index out of bounds or Variable subrange exceeded.                                              |
| 2903a      | Attempts to input the real number <code>—0.0</code> cause programs to enter infinite loops.                                                                                                                                                                                                            |
| 2953       | In certain cases, line numbers are reported incorrectly in the walkback when                                                                                                                                                                                                                           |
| 3019       | <code>%include</code> files are used.                                                                                                                                                                                                                                                                  |
| 3002       | An internal compiler error produces an undeleted temps errors during compilation.                                                                                                                                                                                                                      |
| 3004       | No error message is reported when an otherwise legal comment is not closed at the end of a file.                                                                                                                                                                                                       |
| 3019       | A function whose return type is a packed array of characters generates incorrect stack offsets.                                                                                                                                                                                                        |
| Unassigned | The <code>\$nowalkback</code> embedded option fails to turn off run-time error walkbacks. For modules compiled with <code>-nowalkback</code> specified on the command line, attempts to use the <code>\$walkback/ \$nowalkback</code> embedded options cause incorrect procedure names to be reported. |
| Unassigned | In certain cases involving non-local <code>goto</code> statements, the compiler generates incorrect embedded code for the Profiler. Programs compiled with the <code>profile</code> option and affected by the problem fail with the error message Fatal error -- Stack underflow.                     |

## Pascal-2 for M686000/UNIX

Version 2.1I, released November 1, corrects the following problems.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                  |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2730          | Specifying a pointer to a file variable in the argument of a <code>rewrite( )</code> statement caused a core dump upon termination of the program.                                                                                                  |
| 2796          | Attempting to use a function that returned a value of a structured type as the argument in a <code>writeln</code> statement caused a program to terminate with a "bus" error.                                                                       |
| 3064          |                                                                                                                                                                                                                                                     |
| 2864          | Certain uses of boolean sets caused addressing problems.                                                                                                                                                                                            |
| 2882          |                                                                                                                                                                                                                                                     |
| 2868          | Certain uses of loophole caused the compiler to hang indefinitely.                                                                                                                                                                                  |
| 2878          | The compiler aborted with a Floating point overflow <code>O</code> message whenever a period was mistakenly substituted for a comma in the type declaration of an array, as in the example type <code>matrix = array [1..10,1..10] of char</code> . |
| 2879          | Code generated by the compiler for the loophole function contained <code>movb</code> instructions where a single <code>movl</code> would be more effective.                                                                                         |
| 2881          | Nesting of more than 15 levels within expressions caused the compiler to generate the message Internal compiler error . . . instead of the appropriate syntax error message.                                                                        |

- |      |                                                                                                                                                                                                                                                         |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2884 | Incorrect subrange checks produced the error message Variable subrange exceeded when variables having a legal value were used as the lower limit in for ... downto ... do statements.                                                                   |
| 2971 | In the definition of certain structured constants, the compiler failed to report the assignment of values outside of a predeclared subrange. Programs attempting to use such constants in comparisons failed with an illegal instruction and core dump. |
| 2973 | Programs that called certain procedures with very large local variables failed with segmentation violations.                                                                                                                                            |
| 2988 | Programs that contained variables requiring more than 32K-byte offsets failed because of incorrect address calculations.                                                                                                                                |

## Known Problems

The following is a list of problems known to exist in Version 2.1I of

**Pascal-2 for M68000/UNIX.**  
All are planned for correction

in a future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                       |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2241          | The compiler produces incorrect code for set assignments when the set is a record field of a packed array of packed records. In these cases, Pascal-2's optimizations cause the compiler to generate byte rather than word instructions and lead to the incorrect results.                                                               |
| 2432          | When you invoke both the <code>—debug</code> and <code>\$nolist</code> compiler options, the compiler currently creates an incomplete listing file. This incomplete listing file causes the Debugger to crash whenever you run the program under <code>pdb</code> .                                                                      |
| 2586          | A mismatch between a type definition and a structured constant definition using that type causes the compiler to crash and produce a core dump.                                                                                                                                                                                          |
| 2731          | When a run-time error is encountered in a Debugger-controlled program, the Debugger is unable to recover the stack frame from which the error occurred and, as a result, it loses access to the program's variables. This condition will be corrected by the introduction of a post-mortem analyzer in a future release.                 |
|               | As a temporary work-around, when you encounter a run-time error during debugging, set a breakpoint at the statement in which the error occurs, then restart the program and run it to that breakpoint. Once you reach the statement, you can accurately probe variables immediately before the point at which the run-time error occurs. |
| 2880          | Several examples of code involving sets implemented with structured constants cause the compiler to enter an infinite loop.                                                                                                                                                                                                              |
| 3193          | Programs fail whenever they contain a procedure whose name is the same as that of the common block containing global variables. This condition occurs most frequently when the <code>\$own</code> option is used in a compilation unit having identical program and procedure names.                                                     |
| Unassigned    | In some cases, the Debugger write command ( <code>W</code> ) fails to write the final field in a record.<br><br>As a work-around, you can view the value of the final field by writing it as a separate variable rather than writing the entire record.                                                                                  |

## Pascal-2 for iAPX 86 Family Processors Running UNIX

Version 2.1D was released  
December 13 for field test. The

following is a list of known problems  
in the software. All are planned

for correction in a  
future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2241          | The compiler produces incorrect code for set assignments when the set is a record field of a packed array of packed records. In these cases, Pascal-2's optimizations cause the compiler to generate byte rather than word instructions and lead to the incorrect results.                                                                                                                                                                                                                                                                                  |
| 2432          | When you invoke both the <code>—debug</code> and <code>\$nolist</code> compiler options, the compiler currently creates an incomplete listing file. This incomplete listing file causes the Debugger to crash whenever you run the program under <code>pdb</code> .                                                                                                                                                                                                                                                                                         |
| 2731          | When a run-time error is encountered in a Debugger-controlled program, the Debugger is unable to recover the stack frame from which the error occurred and, as a result, it loses access to the program's variables.<br><br>As a work-around, when you encounter a run-time error during debugging, set a breakpoint at the statement in which the error occurs, then restart the program and run it to that breakpoint. Once you reach the statement, you can accurately probe variables immediately before the point at which the run-time error appears. |
| 2882          | The compiler aborts on certain code involving sets implemented with structured constants.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Unassigned    | The Pascal-2 compiler generates incorrect 80287 mnemonics when loading an integer variable into the <code>fpp</code> registers and when storing the contents of an <code>fpp</code> register in an integer variable. Currently, the generated assembly code does not distinguish between integer and floating point moves. However, the object code produced by the compiler is correct.                                                                                                                                                                    |
| Unassigned    | The <code>short</code> option stores all integers in 16-bit format. However, when integer arithmetic is performed, all <code>short</code> integers are expanded to 32-bits for the operation, then returned to 16-bit format for storing.                                                                                                                                                                                                                                                                                                                   |
| Unassigned    | In some cases, the Debugger write command ( <code>W</code> ) fails to write the final field in a record.<br><br>As a work-around, you can view the value of the final field by writing it as a separate variable rather than writing the entire record.                                                                                                                                                                                                                                                                                                     |

## Pascal-2 for M68000/VERSAdos

The following problems are verified  
for the Pascal-2 compiler running

under VERSAdos. All are planned  
for correction in a

future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                          |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3181          | Executing programs don't respond to the <code>break</code> key.                                                                                                                                                                                                             |
| 3133          | The user manual is unclear in its description of how a <code>READ</code> statement treats an input string shorter than the declared length of the input variable being filled. The manual does not clearly state that single-character I/O is not supported under VERSAdos. |

- |      |                                                                                                             |
|------|-------------------------------------------------------------------------------------------------------------|
| 3163 | A one-byte for loop index is pushed on the stack as a byte, but popped as a word-length quantity.           |
| 3187 | In writing strings to a file, the size limitation of the output buffer results in an invalid record length. |
| 3200 | The documentation does not indicate that the /list and /errors switches are incompatible.                   |
| 3203 | Inefficient code is generated in a loop with 2 pointers and index arithmetic.                               |

## Pascal-2 Cross-Development System

Version 2.0Q for VMS to M68000, released December 23, corrects the following problems.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                    |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1707          | Failure to initialize a register pointing to a packed array of boolean. The register is now properly initialized.                                                                                                                                     |
| 2034          | An origin address value was incorrectly initialized. The address is now initialized correctly.                                                                                                                                                        |
| 2130          | The compiler generated incorrect code when a for loop's starting and ending values were of the form integer div constant. Correct code is now generated.                                                                                              |
| 2132          | The compiler crashed when an undefined symbol was encountered in a two-dimensional structured constant. The compiler now issues an error message.                                                                                                     |
| 2135          | The compiler crashed with an access violation when compiling a duplicate definition of a structured constant. An error message is now issued.                                                                                                         |
| 2205          | Structured constants were stored with incorrect byte order. Byte order is now correct.                                                                                                                                                                |
| 2718          |                                                                                                                                                                                                                                                       |
| 2819          |                                                                                                                                                                                                                                                       |
| 2344          | A call to ref in a structured constant declaration, which is illegal, went unreported. The compiler now issues the error message bad constant.                                                                                                        |
| 2503          | The loophole function generated odd address traps and failed to properly convert a nil pointer to an integer. Loophole now handles both of these cases correctly.                                                                                     |
| 2508          |                                                                                                                                                                                                                                                       |
| 2509          | Underflow caused arithmetic operations on real quantities to abort or to return incorrect results. Now, the quantity is set to zero and execution continues.                                                                                          |
| 2571          |                                                                                                                                                                                                                                                       |
| 2603          |                                                                                                                                                                                                                                                       |
| 2740          | Compilation failed on code containing a call to an external procedure where one parameter was a pass-by-value real or an integer greater than 2. Compilation now proceeds correctly.                                                                  |
| 2753          |                                                                                                                                                                                                                                                       |
| 2752          | The compiler generated a short (word) instruction to access a long word field of a record structure. The proper instruction is now generated.                                                                                                         |
| 2818          | A call to the sqr function with a variable of an integer subrange type caused the compiler to abort with consistency check errors. Integer subranges are now handled properly.                                                                        |
| 2832          | A function call in which one parameter is an expression caused the compiler to abort with an undeleted temps error. The compilation now proceeds to completion.                                                                                       |
| Unassigned    | The support library expected the EXITST procedure to issue a 16-bit status code, when it actually issued a 32-bit code. This caused problems in multi-task programs that passed control via EXITST. The library now correctly expects a 32-bit value. |



## Known Problems

The following is a list of known problems in V2.0Q Pascal-2 for Cross

Development System. All are planned for correction

in a future release.

| <u>Number</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1427          | When a variable declared with <code>origin</code> is used as a <code>for</code> loop index, the specified memory location is not incremented.                                                                                                                                                                                                                                                                                                                                                                              |
| 1710          | The <code>size</code> function returns an incorrect value for a nested packed record of <code>byte</code> .                                                                                                                                                                                                                                                                                                                                                                                                                |
| 1926          | The <code>mod</code> function returns a negative result when passed a negative dividend. This "incorrect" result is intentional in order to maintain consistency across the entire family of 2.0Q products. Version 2.1 will adhere to the standard by always returning a positive result. The work-around for the current implementation is to test for a negative result, adding in the divisor if necessary. (This strategy assumes a positive divisor. A negative divisor is illegal in a <code>mod</code> operation.) |
| 2131          | Maximum allowable size for a packed array of <code>word</code> should be, but is not, exactly twice the maximum allowed for a packed array of <code>byte</code> .                                                                                                                                                                                                                                                                                                                                                          |
| 2226          | A program in which an element of a structured constant is used in a computed array index may cause the compiler to abort with an undeleted temps error.                                                                                                                                                                                                                                                                                                                                                                    |
| 2181<br>2629  | The compiler opens <code>%include</code> files and its own temporary files in protected mode, thus denying other compilations access to these files. This is a restriction imposed by the compiler used to build the VMS/M68000 cross-compiler and will be corrected in Version 2.1.                                                                                                                                                                                                                                       |
| 2377<br>2560  | Errors in <code>%include</code> files are pointed to incorrectly or are diagnosed incorrectly.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 2136          | The compiler fails to detect a constant having a value that exceeds <code>maxint</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 2582          | Deeply nested <code>for</code> loops may cause a program to abort at run-time.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 3101          | The compiler generates superfluous logical shift instructions when initializing an empty set.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 3127<br>3137  | An <code>if</code> statement within a <code>case</code> construct occasionally causes a stack misalignment.                                                                                                                                                                                                                                                                                                                                                                                                                |
| 3165          | A program containing a packed record of booleans causes the VMS/M68000 cross-compiler to abort with an internal consistency error.                                                                                                                                                                                                                                                                                                                                                                                         |
| 3170          | The VMS/M68000 cross-compiler fails to allocate space on the stack for a function return value.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 3174          | When the embedded <code>\$list</code> switch is changed from 'on' to 'off' more than 19 times in a program, the VMS/M68000 cross-compiler sets the switch permanently to 'on'.                                                                                                                                                                                                                                                                                                                                             |
| 3184          | The RSX/M68000 cross-compiler produces an incorrect output format for negative hexadecimal integers.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 3197          | The VMS/M68000 cross-compiler generates word moves to odd boundaries, which causes the program to abort at run-time.                                                                                                                                                                                                                                                                                                                                                                                                       |
| 3199          | The VMS/M68000 cross-compiler erroneously traps division by a nonzero, denormalized number as an attempt to divide by zero.                                                                                                                                                                                                                                                                                                                                                                                                |
| 3201          | The VMS/M68000 cross-compiler generates an incorrect value when an integer value of <code>-maxint-1</code> is assigned to a variable of type <code>real</code> .                                                                                                                                                                                                                                                                                                                                                           |

- |      |                                                                                                                                                                |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3209 | In compiling structured constants, the VMS/M68000 cross-compiler reverses the order of the bytes representing real values.                                     |
| 3211 | The VMS/M68000 cross-compiler aborts with a %JBC-Network Protocol Error when it encounters a large number of errors in a source file.                          |
| 3224 | The compiler generates illegal instructions for the loophole function.                                                                                         |
| 3238 | There is no documentation in the Pascal-2 Cross-Development System User's Manual regarding the library's limitation of truncating symbols to eight characters. |

## Oregon Linker/Assembler

Linker Assembler was released December 23. This version of the cross-assembler includes the following changes:

| <u>Number</u> | <u>Description</u>                                                                                                                                                                     |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unassigned    | The \$ is now recognized as a legal file specification character. The cross-assembler no longer loops indefinitely when the \$ appears as the first character of a file specification. |
| Unassigned    | The cross-assembler produced an incorrect bit pattern for exg instructions. The cross-assembler now processes them correctly.                                                          |
| Unassigned    | Certain complex expressions failed. The cross-assembler now processes them correctly.                                                                                                  |
| Unassigned    | Esd records were not generated for empty sections. The cross-assembler now generates them correctly.                                                                                   |
| Unassigned    | The assembler now generates object code for data declarations found at the end of a file.                                                                                              |
| Unassigned    | The cross-assembler now emits correct bit patterns for operands. Previously, the high order bit was always cleared.                                                                    |
| Unassigned    | The 32-bit multiply routine for M68000 targets now functions correctly.                                                                                                                |

Version 1.0B of the cross-linker includes the following corrections:

|            |                                                                                                                                                                                     |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unassigned | The \$ is now recognized as a legal file specification character. The cross-linker no longer loops indefinitely when the \$ appears as the first character of a file specification. |
| Unassigned | Use of the /cross switch no longer crashes the linker when no external symbols are defined.                                                                                         |
| Unassigned | Using more than 15 different section specifications no longer causes the high merge order sorts to fail.                                                                            |
| Unassigned | The linker no longer loops when the /cross switch is used.                                                                                                                          |
| Unassigned | The link map now displays segments in the correct order with the correct section boundaries.                                                                                        |
| Unassigned | The linker no longer aborts when given a full VMS file specification as an included object module.                                                                                  |
| Unassigned | Certain symbol combinations no longer cause the linker to abort with run-time errors such as Array subscript out of bounds, or Attempted reference to invalid pointer.              |

Unassigned

Multi-segment input no longer causes the linker to abort with a nil reference. Previously, the cross-linker failed to read certain relocatable object files. The result was an S-record file whose load addresses were consecutive integers: e.g., 0, 1, 2, 3,...

Unassigned

The cross-linker now processes certain symbol combinations correctly. Previously, the linker reported internal bad name format for correct combinations of symbols.

## Known Problems

Version 1.0B of the cross-assembler has no known problems; the cross-

linker has one outstanding bug, which is planned for

correction in a future release.

### Number

Unassigned

### Description

The cross-linker does not place the form feed correctly in multiple page cross-reference listings.

## SourceTools

The following problems are verified for SourceTools V1.0B, and are planned for correction in a future release.

### Number

2854

### Description

SourceCon, when reading keyword and deltaword values specified without parentheses in the command line, fails to detect the end of the value string. The entire remainder of the line is taken as the symbol value.

3218

When the command for a target is missing, MAKE points the error message to the dependency on the following line.

**Pascal**  
**NEWSLETTER**  
OREGON SOFTWARE

6915 SW Macadam Avenue  
Portland, Oregon 97219



John Peterson  
Apt #714  
130 South, 13th East  
Salt Lake City, UT, 84102



# NEWSLETTER

WINTER/SPRING 1987

TECHNICAL NEWSLETTER

NUMBER 12

PAGE 1

## Standardization issues confront potential Modula-2 implementors

by  
David M. Barnes

Modula-2 is a language created by Professor Niklaus Wirth, the designer of Pascal. Modula (as it is conventionally known) is similar to Pascal in appearance but offers some powerful capabilities not provided by standard Pascal. When he designed Modula-2, Professor Wirth provided a consistent approach to filling in many of the semantic gaps in standard Pascal. The name, for example, is derived from Modula's scheme for separate compilation units, or "modules." Separate compilation is an extension that most Pascal implementors have provided, each in its own way. In Modula-2, the division of separate compilation units is standard because it is specified in Professor Wirth's language definition. Similarly, Modula provides extensions downward to interface with the operating system, and upward to interface with applications.

Wirth's book *Programming in Modula-2*, third edition (affectionately known as PIM-2.3) sets forth his definition of the Modula-2 language. PIM-2.3 serves two categories of readers: users and implementors. The demands of the two types of readers differ. Users need to understand the fundamentals of the language: the data and control constructs it offers and the syntactic rules for writing correct Modula-2 code. Implementors, too, must understand the language fundamentals and they need additional details regarding the fine points of the language's behavior under all reasonable conditions. Wirth did not provide this level of detail; PIM-2.3 is primarily user-oriented and leaves much to the implementor's intuition.

Professor Wirth did not provide implementation-oriented details for two reasons. First, such detail necessarily dwells a great deal on the handling of incorrect syntax and of absurdly extreme examples. Most users would rather not be burdened with such discussions but implementors must know how to deal with such cases. Second, in many language areas, such as I/O, no clear "best" paradigm has emerged. Professor Wirth chose to give implementors the freedom

to exercise their creativity in these areas. Presumably when a clearly superior approach emerges, he will incorporate it into Modula-2 or some future language design. Professor Wirth has implemented the language himself in more than one environment, and he published his results for other implementors to use. He can certainly be excused for omitting the messy details from PIM-2.3. Nevertheless, the need remains for a clear set of implementation standards in Modula.

The British Standards Institution initiated its Modula-2 standardization working group in 1985 to fill in the important details that Professor Wirth did not address in his book. The committee is composed of respected computer language experts from both sides of the Atlantic, and from both industry and academia. One member is Oregon Software's Compiler Group Manager, Randy Bush. The working group maintains a steady flow of literature on a wide variety of topics. There is a great deal of work to be done, and it is unlikely that an international standard will emerge this year. In the meantime, Modula-2 implementors must resolve such problems for themselves.

This article describes some of the issues that implementors face and outlines some of the possible approaches that have emerged. Two representative topics are string handling and the CARDINAL data type.

### Strings less restricted

String handling is a topic near and dear to the hearts of Pascal programmers, and is widely regarded as one of the "semantic gaps" left in standard Pascal. The standard's requirement that a string literal be assigned only to a PACKED ARRAY OF CHAR of identical length renders string-handling code both awkward and inefficient. Most implementors have addressed the shortcoming by providing supplementary libraries of string-handling routines or by adding a STRING data type to the language.

Professor Wirth, in PIM-2.3, does not provide a STRING data type. He does, however, relax the requirement that strings in assignments be of equal

length. The target string need only be long enough to accommodate the string literal it is to hold. Thus, the following assignment is legal in Modula-2:

```
VAR
  Target : ARRAY [0..10] OF CHAR;
...
  Target := "ABC";
```

A second, more subtle, feature completes the PIM-2.3 string-handling scheme: a single character, normally regarded as type CHAR, is assignment-compatible with variables of type ARRAY OF CHAR. Following the above example, the assignment `Target := "A"` is legal. To users accustomed to STRING data types, this case may seem obvious. To an implementor, it constitutes a significant deviation from the strict typing requirements of the Modula-2 language.

These two string-handling features provide much of the functionality required for character string manipulation without introducing an additional STRING data type.

Some implementors and some members of the BSI working group regard Professor Wirth's solution as a satisfactory remedy to the string-handling shortcomings of standard Pascal. Others find the solution inadequate; the language does not allow string comparison (`IF string1 = string2 ...`) and provides no concatenation operator. Both of these functions can be provided in the form of library routines, but many people would prefer to augment the language with a STRING data type and a full complement of string operators. The BSI working group is currently considering both points of view.

*Continued on Page 4*

### In this issue . . .

|                               |   |
|-------------------------------|---|
| President's message . . . . . | 2 |
| Bulletin board . . . . .      | 2 |
| Technical support . . . . .   | 3 |
| Current releases . . . . .    | 3 |
| Language survey . . . . .     | 5 |
| DOS expanded memory . . . . . | 5 |



## President's message

We have been very busy at Oregon Software, and I feel that we have accomplished many of our goals for 1986. Major efforts have been directed toward solidifying our current products as well as releasing new ones.

Pascal-2 for MS-DOS became a reality in late 1986 and we are now able to offer PC users the same high quality and performance that our current DEC and Motorola users enjoy. We expect widespread acceptance of the MS-DOS product by those Pascal users who have exceeded the capabilities of other vendors' products. In addition, we have made available cross-compiler products for new host-target combinations such as VAX to MS-DOS, MS-DOS to 68000, or MS-DOS to VAX. We have found that customer interest in these products goes beyond the "sophisticated user" group. Hosted on MS-DOS, the cross-compiler is bound to become a much more useful tool because the power of MS-DOS systems is being expanded by introduction of the new 80386 machines and by use of accelerator boards for 80286 systems. For another more powerful environment, we are now testing our enhanced M68K product: Pascal-2 with 68020/68881 support. This version generates code for both the 68000 and 68020 machines.

Pascal-2/VMS Version 2.11 is clearing field test and will be available in early April. In October, we released Pascal-2

and SourceTools updates for PDP-11/RSX systems. Both releases took longer than we'd anticipated, but the new versions provide capabilities that users requested. The Pascal-2/RSX maintenance release repaired many of the bugs reported by users. The SourceTools/RSX commands UPDSRC and GETSRC can be executed in much less time than previous releases. The forthcoming SourceTools/VMS release contains greater increases in execution speed for UPDSRC and GETSRC. A new method of handling file protections provides increased security and protection of vital data. Also, the longer VMS Version 4 file specifications are now supported and error reporting has been improved throughout.

Oregon Software continues to serve the high-performance needs of users around the world and we are happy to report that our Modula-2 program has begun. Our initial development will take place in the VAX/VMS environment. We plan to decide the most complex design issues we face and have a prototype compiler running by Christmas.

We've placed more emphasis on the process of testing, packaging, and integrating our products. Our engineering reviews now include strict procedural requirements for these activities. Oregon Software's staff now includes a team of software testers who have developed several automated suites of extensive QA tests, ensuring delivery of a better tested product to our customers.

Our Marketing/Sales area has been expanded to include a field support function. In order to provide better and more responsive support capability for our products, we've dedicated a team of support engineers and are setting up an electronic bulletin board to assist in servicing some of the unique time zone problems.

We look forward to providing you with a continued suite of products that improve your productivity and performance. Thank you for your past loyalty and support as well as the excellent word-of-mouth recommendations you make to your colleagues.

*W. W. Borowsky*

W. Walt Borowsky  
President and CEO  
Oregon Software, Inc.

OREGON SOFTWARE

## NEWSLETTER

6915 SW Macadam Avenue  
Portland, Oregon 97219

David Spencer, editor  
David Barnes, Norm Hartman, staff writers  
Betsy Slonaker, production assistant

The Pascal Newsletter is published regularly by Oregon Software, Inc., 6915 SW Macadam Ave., Portland, OR 97219; (503) 245-2202. Subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor.

Copyright © 1986 by Oregon Software, Inc.  
ALL RIGHTS RESERVED.

The following are trademarks: Oregon Software, Pascal-1, Pascal-2, SourceTools, Oregon Linker/Assembler, and Pascal Newsletter, Oregon Software, Inc. DEC, RSX, microRSX, RSTS, RT-11, P/OS, PDP-11, VMS, ULTRIX, VAX, and microVAX, Digital Equipment Corporation. UNIX, AT&T Bell Laboratories, Incorporated. M68000 and VERSAdos, Motorola, Incorporated. XENIX and MS-DOS, Microsoft Corporation. IBM, International Business Machines Corporation. Intel, Intel Corporation.  
Printed in USA

## Support bulletin board goes on line

by  
Jan de Rie

Oregon Software recently installed a bulletin board as a support tool for the MS-DOS Pascal-2 compiler. Anyone with a 300, 1200 or 2400 baud modem and a terminal can access the board by dialing (503)244-6102. File transfer is supported for various protocols, including XModem and Kermit. The bulletin board can be used to retrieve and to send information, messages and files.

A very important feature of the bulletin board is the availability of in-depth application notes on how to use Pascal-2 in specific areas. Current application notes cover floating-point emulation, writing interrupt routines in Pascal-2, and using DOS file handles. Other application notes will follow and users are invited to submit application notes. Another feature will be the availability of SEDIT, the Oregon

Software stream editor. This program allows us to post on the bulletin board edit scripts that convert programs—TURBO Pascal sources, for instance—for compilation by Pascal-2. Source code of general interest, such as an updated version of P2HANDLE.PAS, will be posted on the board. Known problems and work-arounds will also be posted. Additionally, we will have information available regarding other Oregon Software products.

It is also possible to send and read messages on the board. This is helpful for more complicated questions and for submitting bugs. The files needed to duplicate the bug can also be transferred to the board. So all MS-DOS Pascal-2 users, call the bulletin board and share your experiences.

# Technical Support: New Friends Solve Old Problems

John P. Rice  
Technical Services Manager

Traditionally, the support technician's role in a software company is to be the first recipient of complaints from angry, frustrated customers. After all, their dialogue centers on a problem: customers rarely call just to chat or exchange pleasantries. From my years of duty on the support lines, I've identified certain guidelines that can alleviate some of the tension and expedite a solution. However, both the customer and the support engineer have to prevent common ground from becoming a battleground.

OSI support staff will initiate the support relationship by calling each new customer soon after the software is purchased. They'll be introducing themselves and offering assistance with any future problem. They'll also want to identify a key person for future contact.

When taking your calls OSI support engineers will, of course, be polite and courteous, trying to empathize with your dilemma. To help you with the problem in the most efficient way, they'll follow these guidelines:

- Ask many relevant questions and document your responses. On many occasions, these questions help you to discover a solution of your own. At least, good questions can help you to understand and pinpoint the problem area.
  - Solicit comments and suggestions. We always appreciate your cooperation. A customer's opinions concerning our products are a valuable resource in the task of fixing system problems and enhancing new releases.
  - Whenever possible, follow-up phone conversations with a note or letter recounting events and stating our understanding of your position.
- Before contacting the software company's technical support group, try to research the topic or problem thoroughly. Collect any relevant listings and document any error messages or other run-time diagnostics. Be prepared to ship or transmit this information along with files or programs stored on compatible media (disks, tapes, etc.).
  - Initiate phone conversations by supplying the necessary account and product information:
    1. customer name (company name) and site license number if used
    2. telephone contact and number.
    3. exact product description, including version number, and the environment (hardware and operating system at least) in which the product is used.
    4. any modifications or options used.
    5. date the product was purchased and the expiration date of any maintenance agreement in effect.
  - Try to be realistic concerning the timeframe required for response. If immediate response is needed, say so. However, less pressured responses will be greatly appreciated by the support technician. Also, when every customer identifies their problem or request as "urgent" or "critical," support groups cannot justly prioritize problem handling.

User groups are an excellent resource for "comparing notes" on a problem or simply getting another knowledgeable opinion. If possible, contact a fellow user of the product involved and ask whether he/she has encountered the same problem

before contacting technical support. Also, more than one customer with the same problem carries a lot of clout in dealing with the software company.

I have established many professional friendships over the years in my dealings with customers. Although occasional disagreements are inevitable in the arena of software support, a little effort to empathize with the other guy will go a long way in helping to resolve problems and establish valuable contacts in the industry.

## Support engineers ready

Two-thirds of the support staff is new. Rita Gorham, formerly the team's administrative aide, is now a sales representative. Senior support Engineer Robert Pawelski is still on the phones daily and is creating a new data base for faster and better tracking of Field Reports. Technical Services Manager **John P. Rice** brings to the group a background in teaching and consulting. This will be the third technical support position John has held in software companies. **John Langland**, a Technical Support Engineer, has a BS in Computer Science from Portland State University. He's focussed on supporting Pascal-2/MS-DOS right now and is presently engaged in setting up a bulletin board service (BBS) where customers can find support information and may trade messages. John's programming background is in UNIX and C.

When no immediate resolution is available, the support technician will indicate a specific time in which he'll respond to you. The reply, "I don't know but I'll find out," is usually an acceptable response. We know that you can easily relate to the "unknown" aspects of problem solving. Even if no response or solution has been produced, a customer deserves to be notified that his problem is, at least, being given attention.

For customers, I suggest the following approach to support requests:

## Recent and Upcoming Releases

| Products                        | Field Test Release | General Release |
|---------------------------------|--------------------|-----------------|
| Pascal-2 VAX/VMS to 68000 V2.1I |                    | Jan 87          |
| SourceTools RSX V1.1A           | Jan 87             | Mar 87          |
| Pascal-2 VAX/VMS V2.1I          | Jan 86             | Mar 87          |
| Pascal-2 VAX/UNIX V2.1I         |                    | Apr 87          |
| Pascal-2 68000/VERSAdos V2.1J   |                    | Apr 87          |
| Oregon Linker/Assembler V1.1A   | Mar 87             | Apr 87          |
| Pascal-2 68000/UNIX V2.1J       |                    | May 87          |
| Pascal-2 PDP-11/RSTS V2.1E      |                    | May 87          |
| Pascal-2 VAX/VMS to 68000 V2.1J |                    | May 87          |
| Pascal-2 68000/VERSAdos V2.1K   |                    | Sep 87          |

Continued from Page 1

## CARDINAL type debated

The CARDINAL data type described by Professor Wirth in PIM-2.3 denotes a range of positive whole numbers whose maximum value is defined by the number of bits in the host machine's natural word size. The INTEGER data type, by contrast, must reserve one bit to carry sign information. Thus, the largest CARDINAL value is greater than the largest INTEGER value:  $\text{MAX}(\text{CARDINAL}) > \text{MAX}(\text{INTEGER})$ . To the reader's eye, a CARDINAL value is indistinguishable from a positive integer. For example, what is the data type of the constant Answer in this declaration?

```
CONST
  Answer = 42;
```

Many users would respond "Who cares?" Implementors care. In a strongly typed language such as Modula-2, the data type of each operand in an expression must be known to the compiler. In an assignment statement in which the left-hand variable is of type CARDINAL, the right-hand expression must evaluate to type CARDINAL. The following example shows two assignment statements. The first is legal because 2 and 1 are CARDINAL. The second is illegal by the rules of PIM-2.3 because  $-1$  is an INTEGER.

```
VAR
  C : CARDINAL;
...
C := 2 - 1; ----- legal
...
C := 1 - 2; ----- illegal
```

In the next example, the CARDINAL return value of the standard function ORD must be converted to type INTEGER to accommodate a possible negative result:

```
TYPE
  Weekday = (Sunday, Monday,
             Tuesday, Wednesday, Thursday,
             Friday, Saturday);
VAR
  I : INTEGER;
  A, B : Weekday;
...
I := INTEGER( ORD(A) )
    - INTEGER( ORD(B) );
```

Subtler cases arise when left-to-right evaluation of the expression requires computation of an intermediate value.

For example, the following statement is not legal because  $3 - 4$  produces a negative intermediate value:

```
C := 3 - 4 + 2;
```

The primary purpose of CARDINAL was to allow access to the sign bit on 16-bit hosts. On 32-bit hosts (in keeping with Professor Wirth's implementation of the language on the Ceres NS32000-based machine), access to the full 32-bit storage word is not generally regarded as critical enough to justify a special whole-number data type. The trend is to relax the boundary between the two data types and treat CARDINAL as a subrange of INTEGER. Compilers adhering to this concept contain the implicit declaration:

```
TYPE
  CARDINAL = [0..MAX(INTEGER)];
```

Thus, the example  $C := 3 - 4 + 2$  becomes a legal expression: the negative intermediate value is of type INTEGER and the result type is a legal member of the CARDINAL subrange.

Implementors must choose whether to uphold the integrity of CARDINAL as a distinct data type or to allow it to be defined as a subrange of type INTEGER.

## Adopted or de facto standard

Strings and data types represent only two of the many issues on which a standards committee must rule if their work is to be useful as a set of across-the-board implementation guidelines. Unfortunately, the issues facing the BSI working group seem to multiply faster than they can be resolved. Until the BSI standard emerges, implementors must resolve such issues themselves.

One course chosen by some implementors is to create a Modula-2 compiler that matches that described in *Programming in Modula-2*. This is a good stopgap measure but the PIM-2.3 version of the language is hindered to some degree by its 16-bit orientation and the book's lack of implementation-level detail. In addition, a PIM-2.3 compiler is unlikely to match the BSI standard when it eventually emerges. Another possible path is to implement a Modula-2 language that resembles the PIM-2.3 compiler, but with extensions and changes that suit the taste of the implementor. Again, the problem of non-conformance to the eventual standard is a risk.

Both of these possibilities are better, though, than the third alternative, which is to defer implementing the language until the standard has been defined.

Modula is a powerful and significant language, and deserves to be widely used as soon as possible. Implementors who forge ahead with their own resolutions to the issues described in this article may, in fact, establish *de facto* standards that the committee can incorporate into their work.

## Bits & Bytes

This column contains brief announcements and messages. We invite users to send us their favorite tips, magic commands, and dirty-but-useful tricks.

## New type for OSI manuals

OSI's Technical Publications team has taken two steps to improve the quality of the print in manuals. First, we've acquired a new laser printer, with resolution of 300 dots per inch to replace the 240 dpi model we've been using. The new laser printer will be used for preliminary copies of manuals and for release notes. Secondly, we've found two local printers who can do typesetting from our TeX sources. Final versions of our manuals will be produced at a resolution of 1200 dpi. No longer will our readers have to wonder whether the black rectangle is a capital N or a cursor symbol; the at-sign (@) will begin to look like its old self again.

— David Spencer

## Accessing command-line arguments under UNIX

Pascal-2/UNIX provides a standard command-line interface called `getparam` which returns the actual *i*th command-line argument. Argument numbers start from 0 and end with *argument-count* - 1 where *argument-count* is the total number of command-line arguments, or *argc* to those of you who speak C. An argument length of -1 is returned if the argument number is invalid. Typically, an argument number greater than *argument-count* - 1 causes this error.

The "getparam" interface uses another library function, `u_getarg`, which returns a pointer to a string of null-terminated characters (mentioned in the last issue's article on host integration).

The sample program shows you how `getparam` may be used.

— Carl Ellis



Figure 1: Use of the GETPARAM Procedure

```

procedure getparam(argument_number: integer;
  var argument_str: packed array [1..h: integer] of char;
  var argument_len: integer); external;

const
  max_arglen = 256;

type
  arg_str = packed array [1..max_arglen] of char;

var
  i, len: integer;
  arg: arg_str;

begin
  len := 0;
  i := 0;

  while (len >= 0) do
    begin
      getparam(i, arg, len);
      if len >= 0 then
        begin
          write('argument ', i: 1, ' = ', arg:len, ', and is ', len: - 1);
          writeln(' characters long');
          end;
          i := i + 1;
        end;
      end;

end.

```

## Oregon Linker/Assembler offers increased power

An important part of our complete cross-assembler product, Version 1.1A OL/A allows users to take advantage of greater capabilities offered by the Motorola 68010, 68020 and 68881 processors. In addition, you may use position-independent code.

To complement its new strength, the user manual has been completely rewritten to clarify previous information and expand the section on the linker.

## Which languages do you want from Oregon Software?

We plan to offer Modula-2 late in 1987 but are considering others. We're particularly interested in hearing what hosts, targets, performance, price, or other product features would cause you to purchase a language product from us. Languages under consideration include C, C++, Fortran, and Ada. Please call 800-874-8501 for "Language Survey" to register your comments or write us at:

Oregon Software  
ATT: Language Survey  
6915 S.W. Macadam Ave.  
Portland, OR 97219, USA.

# Accessing expanded memory with DOS function calls

"How do I access memory above the 640K MS-DOS limit?" has been one of the frequent questions asked of Oregon Software's support staff recently. The answer is straightforward, but first let's make clear a distinction between "extended" memory and "expanded" memory. The two approaches are commonly confused.

Extended memory is the memory between 1MB and 16MB, accessible in the AT's protected address mode. You can access such memory from Pascal-2 but your program incurs the overhead of the protected mode access. Currently XENIX is the only operating system which supports this mode, although some software (a RAM disk, for instance) makes use of extended memory under MS-DOS.

Expanded memory is paged memory that is implemented through bank-switching and organized in pages of 16KB with four contiguous pages making a 64KB page frame. A base page frame is

assigned in user memory (the first 640KB) during configuration of the board.

Programs access expanded memory through frame registers which are implemented as I/O ports. Your program uses DOS function calls accessed through INT 67 to communicate with the EMM device driver which is installed at system boot time.

The procedure `emm_call` in Figure 2 illustrates the syntax for register access and the use of the mechanism which provides access to interrupts. The OSI bulletin board contains a full source listing of Peter Handsman's program `EMS.PAS`, which executes the "Sieve of Eratosthenes" in expanded memory.

Figure 2: Procedure to Access Expanded Memory

```

Procedure emm_call(var regsvar:regs; ah:integer);
{ This procedure makes a call to the emm device and executes }
{ the function specified in the ah parameter... also it calls }
{ the error handler if the emm manager returns an error message. }

begin
  regsvar.w.ax := ah*16#100;
  p_int86(16#67, regsvar, regsvar);
  if hi(regsvar.w.ax) <> 0 then error_handler(hi(regsvar.w.ax));
end; {emm_call}

```

## OSI engineering team gains depth

**Don Williams** is the new Director of Engineering at OSI. He has held similar positions at N-DEX Labs and Tektronix Inc. Don's professional interests include software engineering methodology and the effective use of personal computers.

**Randy Bush** is Manager of the Compiler Group. Randy established his Modula-2 expertise as a member of Pacific Systems Group and as an independent consultant. He is a member of the British Standards Institution's Modula-2 Standard Committee.

**John P. Rice** is our Technical Services Manager. In addition to his teaching and consulting background, this will be the third position John has held in Technical Support for software companies.

Junior Software Engineer **Loc Nguyen**, a recent graduate of Portland State University, is making good use of his academic Pascal experience by working on the MS-DOS support library.

**Darrel Schneider** is a Software Technician in our Software Tools group. Darrel will soon graduate from Portland State University with a BS in computer science. He is experienced with 68000 systems and is maintaining the SourceTools program and OL/A, the Oregon Linker/Assembler program.

**Janine Harris** is an Administrative Assistant in our Engineering Department. Janine comes to us from Denver, Colorado where she earned her degree in music education. She was employed by DEC for 5 years as a Customer Response Representative and as Administrative Secretary for Software Services. Janine takes care of administrative details of the department, such as scheduling, production progress tracking, and special projects for programmers and software engineers.

**Norm Hartman** is a writer on our Technical Publications team. He has a two-year certificate in technical writing from Portland Community College. Before joining OSI, Norm worked as a free-lance programmer and technical writer for several years.

**Kim Barrett** is a Software Technician. Kim has a BS in Computer Science from Portland State University and a certificate in graphic reproduction from Portland Community College. Before coming to Oregon Software, she worked at Oregon Health Sciences Center and in photography. Kim's duties include building Pascal-2/PDP-11 releases and trouble-shooting the RSX support library.

**Carl Ellis** is a Software Technician. Originally from Alaska, Carl is finishing his BS in computer science at Portland State University. Carl packages the

Pascal-2 releases for M68000 processors and he supports the development team for UNIX and VERSAdos products.

**John Davis** is a Software Technician in the Software Evaluation team. John was born in Oregon, and educated in Oregon and Canada. He attended the University of Oregon and Mt. Angel, majoring in dramatics, and pursued a career in theater for ten years. A graduate of Portland Community College's software technician program, John tracks down bugs and develops programs to verify that products meet their specifications.

**John Langland** is a Technical Support Engineer. John has a BS in computer science from Portland State University. His current specialty is MS-DOS support, but he will be branching out to support other systems. He is presently engaged in setting up a BBS for our customers where they can look for support and trade information and messages. John's programming background is in UNIX and C.

Software Intern **Garret Hade** is a Portland State University student who helps test our Motorola 68000-based product line.

**Thuy Nguyen** is a senior in computer science at Portland State. Previously, she's worked as a programmer for the U.S. Army Corps of Engineers. As a Software Intern, she's helping with the 68000 project; her future interests are hardware engineering, graphics, and artificial intelligence.

## Sales staff expands

**Vince Jones** has been on board since last July as our Director of Marketing and Sales. He came to OSI from Communication Dynamics with a strong background in marketing and experience in personal computers and graphics applications. Vince brings to software a portfolio of marketing concepts developed in other, more traditional, industries. He applied these skills to opening distribution channels for our MS-DOS compiler, and is now exploring government markets.

**Gary Snyder** joined OSI last November to telemarket the new MS-DOS product line. He previously worked as a telemarketer for Integrated Measurement Systems and for Teneration Corporation.

**Larry Bell** is a sales representative who came to Oregon Software from the 3M company. Larry's background in computer science and statistical programming gives him a thorough understanding of our users' applications.

**Paul Cormier**, Senior Account Executive/Representative, is our man on the East Coast. Paul's mission is to develop the New England market, getting to know the suppliers and markets in the area. He comes to us with a strong background in management and information science.

## Operations team added

Director of Operations **Bob Martin** manages Oregon Software's day-to-day business affairs, develops growth objectives, monitors budgets, and keeps an eye on overall operational efficiency. Bob came to Oregon Software from N-DEX Labs, where he served as GM and operations manager for three years.

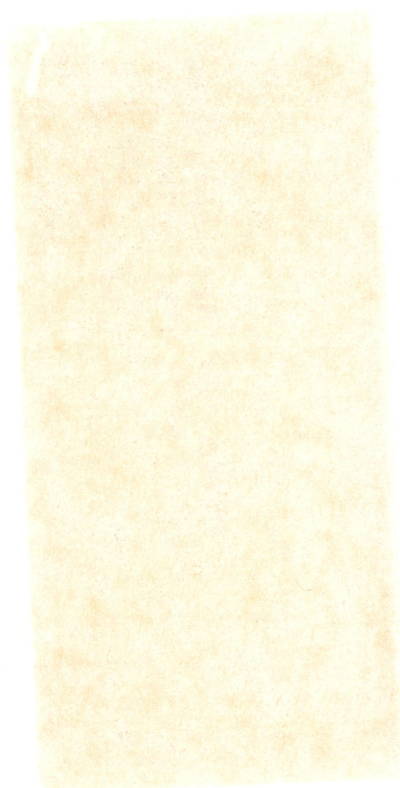
Controller **Cathy Ashcraft** joined OSI in December, bringing with her fifteen years' experience in the accounting field. Cathy's career has taken her to such distant locations as Germany and Alaska, and she has worked in geotechnical, military, and medical fields.

**Collette Smith** is our receptionist. Collette was previously employed as a receptionist by Four A's Temporary Service.

**Karin Mitchell** serves as Marketing Sales Secretary. Karin is originally from Flensburg, West Germany, which lies just south of the border with Denmark, and was educated in Europe. Before coming to Oregon Software, Karin worked for Group 3 Electronics as an executive secretary. Her present job is to support the CEO/President and to work with the Marketing/Sales Director and his staff.

**Andrew Edgar** is the accounts receivable section of our accounting department. Andrew graduated from a four-year program at Portland Community College and is continuing his education at Portland State University. His responsibilities include order entry, collections, receivables, cash receipts, sales analysis and miscellaneous accounting functions.

**Stacey Hoss** is a bookkeeper in the accounting department. She studied accounting in college and was previously employed as Office Manager by Heritage Village Mobile Home Park. Stacey is responsible for accounts payable, general ledger, payroll and miscellaneous accounting tasks.



OREGON SOFTWARE  
**NEWSLETTER**  
TECHNICAL NEWSLETTER

6915 SW Macadam Avenue  
Portland, Oregon 97219



John Peterson  
Apt #714  
130 South, 13th East  
Salt Lake City, UT, 84102